AFIT/DS/ENC/92-1

DEVELOPMENT OF AN ENVIRONMENT FOR SOFTWARE RELIABILITY MODEL SELECTION

DISSERTATION

Salah Amin Elewa
Lieutenant Colonel, Egypt Air Force

AFIT/DS/ENC/92-1

Approved for public release; distribution unlimited

AFIT/DS/ENC/92-1

DEVELOPMENT OF AN ENVIRONMENT FOR SOFTWARE RELIABILITY MODEL SELECTION

DISSERTATION

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Salah Amin Elewa, B.S,M.S.

Lieutenant Colonel, Egypt Air Force

A-1

September 1992

AFIT/DS/ENC/92-1

DEVELOPMENT OF AN ENVIRONMENT FOR SOFTWARE RELIABILITY MODEL SELECTION

Salah Amin Elewa, B.S,M.S.

Lieutenant Colonel, Egypt Air Force

Approved:

Panna B. Nagarsenker — Sept. 15 1992

B. N. Nagaraj — Sept 15, 1992

Henry B. Potoczny — Sept. 15, 1992

[illegible] — September 15, 1992

Brian W. Woodruff — 15 Sept 92

[illegible]

Robert A. Calico

Senior Dean

Preface

The purpose of this study was to design an environment for proper software reliability model selection. Previous investigations have indicated that a model that seems to be the best for one set of data may give very poor results with another set. This is not surprising since the assumptions for each model cannot be satisfied in all situations, due to variations in the development and testing environments for each project. In light of this fact, the idea behind the Proper Model Selection (PMS) environment was to liken a set of collected data to other known sets that previously proved to fit well with a certain software reliability model, or models. Anyone who is interested in determining a proper software reliability model to be used with a set of software failure data should find this environment to be useful.

I must acknowledge special indebtedness to my supervisor Dr. Panna B. Nagarsenker for the wise counseling, encouragement, and ongoing support which made this work, I hope, successful. To Dr. Brahmanand N. Nagarsenker, I pay my sincerest thanks for his vigorous help and continuous support. He has always been generous and encouraging. I am also deeply indebted to Dr. Henry Potoczny for the continuous help throughout the courses and the dissertation research. I would also like to thank Dr. Ben Williams for his enthusiasm, support, and intelligent remarks. Gratitude is also due to Maj. Woodruff for his careful reading of the initial draft and his helpful suggestions. Finally, I wish to express my appreciation to my wife for her patience and encouragement throughout this project.

Salah Amin Elewa

Table of Contents

List of Figures

Figure | Page

List of Tables

Table	Page

AFIT/DS/ENC/92-1

Abstract

An environment was developed for solving the problem of selecting a proper software reliability model for a given set of software failures. The idea behind the environment developed in this dissertation was to liken a collected set of software failure data to a previous one that proved to fit well with a specified software reliability model. Software failures were assumed to have a two-Parameter exponential distribution with unequal type *II* censoring. A test criterion was derived for testing the equality of software failure data sets using the Maximum Likelihood Ratio criterion. The exact distribution of the test criterion was derived. An asymptotic approximation was also obtained and was found to be very close to the exact distribution when the number of failures were more than twenty. Software failure data available from "Data and Analysis Center for Software (DACS)" were used as the initial group of software failure sets. The environment was then applied, for testing the equality of several software failure sets.

DEVELOPMENT OF AN ENVIRONMENT FOR SOFTWARE RELIABILITY MODEL SELECTION

I. INTRODUCTION

Software embedded in microprocessors has been intensively relied on in the past two decades to perform more complex and critical tasks than ever before. Such tasks now include space shuttle programs, weapons coordination, nuclear power plant control, air traffic control, power distribution systems, and medical support systems. Relying on software to perform such mission and life critical tasks, and the sharp increase of software cost relative to that of hardware, has led to the emergence of software reliability field as an important research area. Some progress has been achieved in this field through modern programming techniques, new methods of testing and proof of correctness, and the adoption of fault-tolerant software products. Despite the whole work and research done, a perfect software product is not yet guaranteed. This problem will not be solved in the near future due to the following reasons:

1. Software is created by error-prone humans, and there is no way to prevent programmers from making mistakes,
2. Systems are becoming more complex. This, in turn, will make the task of checking all the paths in a software code a very difficult task. As Dijkstra [25:861] states:

> As long as there were no machines, Programming was no problem at all; when we had a few weak computers, Programming became a mild problem, and now we have gigantic computers, Programming has become an equally gigantic problem.

1.1 Background

Software testing is an important phase in the life cycle of any software product. This importance comes from the fact that whatever software engineering disciplines have been applied to produce a reliable software, the majority of effort and major development costs will be associated with this phase [92:2]. The most important tool in this phase is a software reliability model. Besides measuring reliability of the software, software reliability models can help in the following ways [68:21–22]:

1. Quantitatively evaluating new software engineering disciplines proposed for improving the process of software development,
2. Evaluating the development status during the test phase of a project. A reliability measure, such as current failure intensity, has been found to be much more practical than other traditional methods such as the intuition of the designers or test team, percent of tests completed, or successful completion of critical functional tests,
3. Monitoring the operational performance of software, and controlling the addition of new features and design changes to the software,
4. Enriching insight into the software product and the software development process, which helps in making informed decisions,
5. Application of software reliability models is also useful in future development of other software systems, due to the experiences gained from analyzing the results.

Despite all of the above benefits, many managers abandon the idea of applying software reliability models to their projects. This can mainly be attributed to the problem of finding the proper software reliability model for a given set of failure data. Reasons are given in the next section.

1.2 The Problem of Software Reliability Model Selection

A large number (currently about 100 [26:322]) of software reliability models have been developed within the last two decades. This is supposed to result in some definitive models that can be recommended to potential users. Unfortunately, this is not the case. Among all existing software reliability models, there is no single model that can be trusted to perform well in all situations. The main reasons for this problem are [1:950],[56:172]:

1. Each model can produce a very different answer from the others, when applied to predict the reliability of a software system. Further, a model that seems to be best for one set of data may give very poor results with another set. This is not surprising, since the assumptions for each model cannot be satisfied in all situations, due to variations in development and testing environments for each project,

2. Even for the same project, good performance of a model for a given period of time does not guarantee the same level of prediction for a later period,

3. There is no available priori method for deciding which model is the most suitable for a given set of data,

4. Software failure data reflect the quality of the project under development. Consequently, many organizations consider these data to be classified. This makes validation and refinement of different models a difficult task, due to the shortage of published failure data.

As mentioned before, the above reasons, as well as time and cost constraints, have made many software practitioners avoid incorporating reliability modeling as a management technique. This situation makes it necessary to devote less effort in developing more software reliability models, and concentrate on finding tools for selecting the proper model for a given set of data. This will help in establishing a software reliability theory similar to that of hardware. This dissertation is part of this effort.

1.3 General Approach

From the above discussion, it is clear that a simple and objective method is needed to select the best model for a given set of data. Our approach for solving this problem is to use the same model that proved to be the best for an equal set of data in the past. The main advantage of this method is that one will make use of the previous analysis made when different models were applied with different sets of data, and of different techniques used to measure the predictive quality of software reliability models, with respect to known sets of data. This will be done without evaluating different models, comparing them, and then choosing the best one. This method can simply be stated as *"matching a software reliability model to a given set of failure data"*. This method will provide a strong framework and a useful data base system that results in the user confidence of the reliability calculations he/she makes.

The sequence for solving the problem can be simplified as follows:

1. Initially, the failure data available from Data and Analysis Center for Software (DACS) [64] will be used as a reference, because most of the research done in the past has used them, either for data analysis or for introducing a model,

2. During the test phase of a software product, failure data will be collected. A proper software reliability model will be needed. To achieve this, collected data will be compared with those of DACS to find an equal set to the collected one,

3. If two sets of data are found to be equal, it is reasonable to assume that the same model used in the past and gave good results with one set will do the same with an equal set. The logic behind this method is similar to that of Littlewood [52:142] when he stated that

> if a user knows that the past predictions emanating from a model have been in close accord with actual behavior for a particular data set, then he/she can have confidence in future predictions for *the same* data.

In this research, the term *an equal* will replace the term *the same* which verifies Littlewood's prediction that at some future time, it may be possible to match a software reliability model to a program via the characteristics of that program [52:152].

4. Any new set of data that proves to be in close accord with a certain software reliability model in the future will be added to the initial sets available from DACS.

1.4 Assumptions

1. Knowing that almost all models assume that no failures occur at time t = 0, and the exponential distribution is the most acceptable one in the software reliability field due to the randomness of test [92:62], will make us assume that the times between failures have a two parameter exponential distribution. Proper tests will be applied to make sure that the underlying distribution is exponential,
2. Because this study does not recommend specific software reliability models or present new ones, any software reliability model that proved to be in close accord with actual behavior for a particular data set is assumed to be a valid model,
3. Since the data available from DACS will be used, the following assumptions will also be applicable [64:3–4]:
   (a) Data are assumed to be actual execution time (CPU time) or running clock time (elapsed time from start to end of program execution on a running computer),
   (b) Failures occurring before correcting the responsible fault are not counted.

1.5 Sequence of Presentation

History of software reliability models, some general concepts, and classification of existing software reliability models, with a brief description of some classical models, will be given in Chapter 2.

Previous approaches to the problem of software reliability model selection are given in Chapter 3. A mathematical background is given in Chapter 4. Derivation of the test statistic necessary for comparing two or more sets of software failure data is given in Chapter 5. The Asymptotic approximation of this test statistic for the case of software failure data sets of more than 20 failures is given in Chapter 6. Chapter 7 describes the developed environment, and how to use it for comparing a set of data with those of DACS or any other set of software failure data, along with the results of application on some sets of available data. Finally, conclusions and recommendations for further research are given in Chapter 8.

II. SOFTWARE RELIABILITY MODELS

Software reliability is defined as the probability of failure-free operation of software under test for a specified period of time in specified environments [68:15]. According to this definition, two identical copies of the same software may be different, if they are used under different operational conditions. A software reliability model is the most appropriate tool for measuring the reliability of software programs. Over the last twenty years, a large number of software reliability models have been proposed, studied, and compared.

In this chapter, the different methods for classifying software reliability models are presented and some existing software reliability models are described. Before going into these classifications, the history of software reliability and some general concepts will be presented. This will make understanding the subject much easier.

2.1 History of Software Reliability

The history of software reliability can be found in many sources [24, 68, 82, 84, 92]. The most updated history at this time is that in M. Xie's book [92:9–21], which can be summarized as follows:

2.1.1 Before 1970

- By the development of computer systems in the sixties, software reliability problems arose, but the dividing line between software reliability and hardware reliability was not clear,
- Although there were not many articles devoted directly to software reliability in this period, it can be considered as the initial stage of the explosion of software reliability models in following periods,
- The description of programming errors as a birth-death process was introduced by Hudson [36] in 1967.

2.1.2 1970 - 1979

- Great development were made in this period, and many software reliability models were introduced. The software reliability field changed direction from proof of correctness to stochastic modeling of the failure process and statistical analysis of failure data,
- A major change in the history of software reliability models took place in 1972 when the Jelinski-Moranda (JM) model [38] and the Shooman model [83] were introduced,
- Mills [59] suggested the error seeding method for estimating the software reliability in 1972,
- Using the Bayesian methodology for estimating the time between failures was introduced by Littlewood [55] in 1973,
- The assumption that the software failures process can be modeled as a nonhomogeneous Poisson process was introduced by Schneidewind [80] in 1975,
- Using execution time instead of calendar time for software reliability modeling was introduced by Musa [63] in 1975,
- The idea of using software reliability for deciding when to stop testing and software release policies was introduced by Forman and Singpurwalla [28] in 1977,
- Goel and Okumoto [31] modeled the process of imperfect debugging in 1978,
- A famous model was introduced by Goel and Okumoto (GO Model) [33] in 1979. Many later models using the nonhomogeneous Poisson process were either generalizations or modifications of this model,
- This interesting period ended by the publication of real software failure data by Musa [64].

2.1.3 1980 - 1989

- In this period, most of the models proposed earlier were applied to real data and their drawbacks were identified and discussed. Many other new models were also introduced,

- Musa's failure data made it possible for many researchers to overcome the difficulty of getting relevant software failure data,
- Okumoto and Goel [77] proposed some software release models and studied optimum release time, considering reliability and cost, in 1980,
- Choosing the proper model among different software reliability models has attracted attention, and tools for comparing different models have been widely studied [41, 42, 50, 66],
- Nonhomogeneous Poisson process models have attracted the attention of many software engineers and researchers [30, 76, 94],
- In 1984, Musa and Okumoto [67] suggested a logarithmic Poisson model based on the execution time theory,
- In 1987, M. Xie [91] modified the Jelinski-Moranda (JM) model [38] by assuming that earlier detected faults contribute more to the total failure rate,
- In 1989, Ohba [75] suggested a generalization of existing software reliability models by considering the possibility of imperfect debugging,
- Many other software reliability models were introduced in this period. Most of these models are briefly described in the above mentioned book by M. Xie (see pages 13–19).

2.1.4 1990 - present

- By the end of the eighties, the software reliability field was saturated with software reliability models. Attention now is directed to other related problems such as tools for model selection, multiversion programming, and software fault tolerance modeling,
- Brocklehurst et al [14] presented a technique called model recalibration for improving reliability predictions,

- Many researchers have deeply studied the fault-tolerance technique to increase the reliability of software programs [6, 8, 11].
- Khoshgoftaar [45] presented a technique for proper software reliability model selection using the Akaike Information Criterion. This technique will be explained later in detail. Another method for proper model selection based on check of limits is presented by Huang Xizi [93].
- The reliability modeling of N version programming is studied by Goseva-Popstojanova and Grnarov [34].

2.2 Some General Concepts

Because software reliability is a recently-developed research field, there are many new concepts introduced in the existing literature. There are also some new ambiguous definitions, which need to be clarified.

2.2.1 Software, Hardware & System Reliability: The increasing use of digital techniques in system design led to the manifestation of three important facts [46]:

1. The main part of the system cost is due to the cost of software development and maintenance,
2. Increased delays in system development and production are primarily due to delays in software development schedules,
3. The reduced reliability of a system containing software is mainly due to unreliable embeded software, where errors are detected after the system has been put into use.

These facts made it necessary to direct more research effcrts in the field of estimating the reliability of systems containing software. The reliability of such systems is a combination of hardware, software, and probably some other factors as shown in Figure (2.1). From this figure, it is clear that proper tools for measuring the reliability of both hardware and software are needed for determining the reliability of systems that contain software.

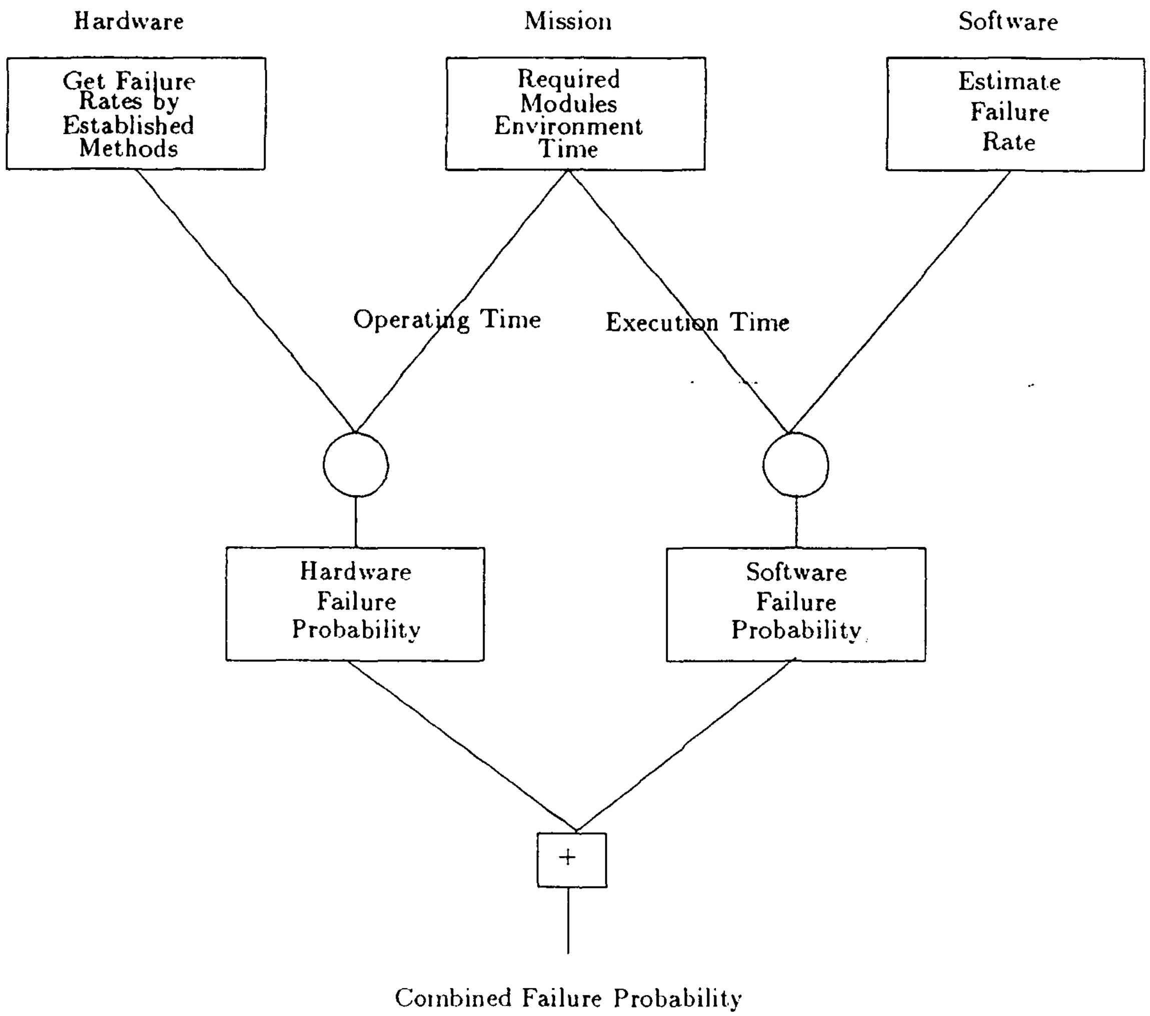

Figure 2.1. Relationship Between Hardware & Software Reliability (McCall[58])

Man used hardware thousands of years ago when the ancient Egyptians used a range of simple tools to build their temples and pyramids and to create their wonderful paintings. On the contrary, computers and software came to existence only in this century. This time gap between software and hardware explains why there has been more research and knowledge in the field of hardware reliability than in that of software reliability. There are some similarities between the two fields. These similarities include defining terms in the same way, which explains why they are sometimes combined to get the system reliability [68:6]. As an example, the hazard rate Z(t) for a software program (***hardware component***) can be defined as the conditional probability that an error (*a failure*) happens in the interval $\{t, t + \Delta t\}$ given that the program (*component*) has not failed up to time t. On the other hand, differences between software and hardware include [24:84],[68:7],[92:7]:

1. Software does not wear out. Rather it becomes obsolete when the environment it was designed for changes,

2. The source of software failures is design faults, while the principal source of hardware failures is physical deterioration,

3. Once a software defect is properly fixed, it is fixed forever, which is not true for hardware,

4. Manufacturing can affect the quality of hardware products, while replication in software products can be performed to very high standards of quality,

5. Software reliability changes mainly during the development phase, while that of hardware usually occurs either in the burn-in phase, or at the end of its useful life phase,

6. Redundancy methodology has no meaning in software reliability and cannot be considered as a tool for improving reliability. The analogous technique in software reliability can be achieved by using multiversion programming,

7. Hardware can be repaired by spare modules, which is not the case for software,

8. Preventive maintenance is very important for hardware, while it has no meaning for software.
9. The hardware reliability field has a well-established mathematical theory but the software reliability field does not.

2.2.2 Faults and Failures:

2.2.2.1 Faults: A program fault (bug) is a defect in the program that causes the program to fail when it is executed under particular conditions. A fault can be viewed either from an absolute point of view or an operational one. From the absolute point of view, a fault is an independent entity that can be defined without reference to failure. In other words, it is a defective, missing or extra instruction or set of instructions. From the operational point of view, the relation between faults and failures is one to one. To compromise between these two views, Musa [68:237] defined the fault as "a defective, missing, or extra instruction or set of related instructions that causes one or more actual or potential failure types". The majority of faults occur during the integration phase, mainly due to the lack of communication between the personnel involved in the software project (programmers, testers, debuggers, ... etc.). Faults can also be in the function of the module itself as a whole, or internal to it (violation of the programming language rules, logical faults, ...etc.).

2.2.2.2 Failures: A program failure is defined as a departure of the output of a program from its requirements. Since failures occur during execution of the program, they are dynamic in nature. Failures are associated with time through the following relations:

1. Time at which a failure took place,
2. Time interval between two successive failures,
3. Total number of failures occurring up to a given time,
4. Total number of failures experienced in a time interval.

From the software reliability point of view, failures occurring as a result of propagation of previous failures are not counted. Failures are said to be of the same type if they occur for the same run type, and are characterized by the same discrepancy, or are due to the same program fault.

2.2.3 Fault Size: There has been much argument about whether faults have an equal effect on the overall reliability of a software program. Some authors, such as Littlewood [51:316], believe that faults do not all have the same effect on reliability. They believe that faults discovered in the earlier stages of testing have higher probability of occurrence. These are considered "big" faults, with a more significant effect on software reliability than those "small" faults inherent in parts of the program that are rarely executed. Thus, the size of a fault is formulated according to its rate of manifestation.

2.2.4 Operational Profile: The execution of a program can be viewed as a single entity, that can last for months or even years for real time systems. However, it will be easier from the software reliability point of view if the program execution is divided into runs.

A *run* is defined as a function that the program performs, such as a transaction in an airline reservation system, or a particular service performed by the operating system for a user. A set of identical runs is called a *run type*. During the test phase, the term *test case* is used instead of the term *run type*. Test cases are chosen to simulate the real operating environments, in order to make sure that the requirements of the customer are correctly met. To achieve this exactly, the tester must be knowledgeable of all the possible run types of the program, and the probability of occurrence of each, which is called the *Operational Profile*.

The operational profile is an extremely useful concept in software testing. However, it may not always be practical to determine the full operational profile because of the large number of run types that can be involved. Thus, it is usually impossible to test all run types of a program. Therefore program failures occur.

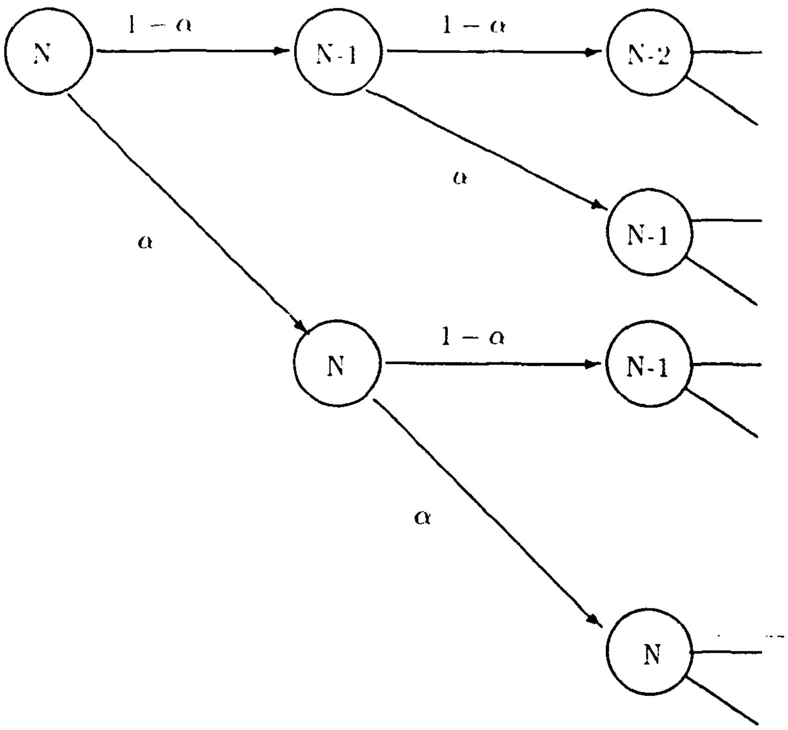

Figure 2.2. An Imperfect Debugging Process [75:240]

2.2.5 Imperfect Debugging: Most software reliability models assume that a fault is completely removed directly after its detection, and that no new faults are introduced (spawned) due to the corrective action. This makes software models mathematically simple, but on the other hand, far from reality. That is mainly due to the fact that "Almost all professional programmers have experienced cases where they fixed one error to create another one"[75:239].

An example of an imperfect debugging process is shown in Figure 2.2. Assuming that the correction action is perfect, the number of faults is reduced by 1, while the number will stay the same or may increase in case of imperfect fixing. If the probability of imperfect debugging is α then the debugging process can be considered as a Markovian process [75:240].

2.3 Software Reliability Model Classifications

Proper classification of software reliability models is necessary to distinguish these models from each other, and to help in proper model selection. The large number of existing software reliability models makes their classification a difficult task. Before going into some classifications, it is useful to know the components of a software reliability model.

Usually a software reliability model involves three main components [1:951]:

1. A conditional probabilistic model specifying the joint distribution of any subset of random variables T_i, conditional on an unknown parameter α (The $T_i's$ are assumed to be independent in all models considered in this chapter),
2. An inference procedure for the value of the unknown parameter α when the actual data are obtained (realization of $T_i's$),
3. A prediction procedure, which combines the above two components to provide predictions for future $T_i's$.

Each model has its own assumptions on the software failure detection rate, the number of faults in the software, and the condition of testing environments. Models can differ also according to the inference procedure and the prediction procedure used. There are several classification systems in the literature for software reliability models. Examples of these systems include:

2.3.1 Classification According to the Model Assumptions: Depending on the assumptions and procedures used for modeling, M. Xie [92:23–29] suggested the following classification system:

2.3.1.1 Markov Models: For a software program under test, the number of faults removed and those remaining at time t are both random values. To explain the reason for this randomness, consider a program that has initially N number of faults. A fault will cause a failure if the program is executed for certain input states. Due to the uncertainty of where faults are located

and the random choice of input states, failures will occur randomly with time. Even when a fault is discovered, there is no certainty about perfect fixing or even the introduction of new faults. The Markov concept is useful in modeling this random behavior. A model is considered to be under this class if its failure-counting process is essentially a Markovian process. Examples of these models are the Jelinski-Moranda Model (JM) [38] and the Schick-Wolverton Model [81].

2.3.1.2 Nonhomogeneous Poisson Process Models: Nonhomogeneous Poisson process (NHPP) modeling is widely used in hardware reliability theory. The same concept is also used for modeling software reliability growth, and many models belong to this class. The failure intensity $\lambda(t)$ for this class of models is

$$\lambda(t) = d\mu(t)/dt,$$

where $\mu(t)$ is the mean value function for the number of detected faults $N(t)$. By using different forms for $\mu(t)$, different NHPP models can be derived. Examples of this class of models are the Goel-Okumoto Model (GO) [33] and the Schneidewind Model [80].

2.3.1.3 Bayesian Models: As mentioned before, at the second phase of software reliability modeling, an inference procedure is used. The classical methods used in most of the software models are the maximum likelihood estimation method and the least square method. The third method is the Bayesian analysis method. The main drawbacks of this method are its complexity and difficulty of application. The basic idea of the method is to identify a prior distribution for the parameter to be estimated (which is the key difficulty in carrying out Bayesian analysis). A posterior distribution is then obtained by multiplying the likelihood function (obtained from the collected data) and the prior distribution. A model is classified under this category if a Bayesian technique is used for parameter estimation. Examples of these models are the Bayesian Jelinski-Moranda Model (BJM) [53] and the Littlewood-Verrall Model (LV) [55].

2.3.1.4 Statistical Data Analysis Models: In this class of models, collected failure data are analyzed using standard statistical methods, such as time series analysis and regression analysis. Software reliability can then be estimated and predicted as a result of these analyses. Examples of these models are the Crow and Singpurwalla Model [19] and the Singpurwalla and Soyer Model [85].

2.3.1.5 Software Metrics Models: Software metrics include software size metrics, software structure metrics, understandability metrics, ..., etc. These metrics are used for predicting the number of faults in a software program. They are useful in earlier stages of software development. The main disadvantage of existing software metrics models is that they predict the number of faults in a program, which is not related to its reliability. More information about these metrics can be found in the papers by Bailey and Dingee [7], Cote [16], Davis and-LeBlanc [20], and Munson and Khoshgoftaar [62].

2.3.1.6 Fault Seeding Models: The basic approach in this class of models is to "seed" a known number of faults in a program, which is assumed to have an unknown number of indigenous faults. Further, it is assumed that the distribution of the seeded faults is the same as that of the indigenous ones. The program is then tested, and the observed numbers of seeded and indigenous faults are used to estimate the fault content in the program prior to seeding. Consequently program reliability and other relevant measures can be estimated. Examples of these models are the Mills hypergeometric model [59] and its modification by Lipow [48].

Some drawbacks of this technique include [53:46]:

1. Increasing the load of testing effort, since the debugger has to manage both original and seeded faults,
2. Seeding faults uniformly in all paths of a software program is always a difficult, if not an impossible task,

3. The total number of faults in a program can not be considered a stand-alone measure of the program's reliability.

2.3.1.7 Input Domain Based Models: The basic approach taken in this class of models is similar to that of fault seeding models. The operational profile concept is used to generate a set of test cases from an input distribution, which is assumed to be a representative of the actual operational profile of the program. Due to the difficulty of obtaining the exact operational profile, the input domain is divided into a set of equivalent classes, each of which is usually associated with a program path. An estimate of program reliability is obtained according to the portion of test cases that cause the program to fail. Examples of this type of models are Brown and Lipow [15] and Nelson [73].

2.3.2 Classification According to the Nature of Failure Process: Under this classification system, Goel [30] classified software reliability models into the following groups:

2.3.2.1 Times Between Failures Models: In this class of models, the time between failures is the process under study. The times between failures are assumed to have a distribution whose parameters will be estimated from the observed values of these times. Estimates of software reliability, mean time to next failure,... etc., are then obtained from the fitted model.

2.3.2.2 Failure Count Models: The main interest in this class of models is the number of program failures in specified time intervals, or those accumulated by a given time t, rather than times between failures. The cumulative number of failures by time t, $N(t)$ is assumed to be the outcome of a random process, which can be completely specified by a discrete distribution function like Poisson or Binomial. Parameters of the models can be estimated from the observed values of failure counts. Again, other estimates of software reliability, mean time to next failure, ... etc., can be obtained from the fitted model.

2.3.2.3 Fault Seeding Models: As described before.

2.3.2.4 Input Domain Based Models: As described before.

2.3.3 Musa-Okumoto Classification: Musa and Okumoto [68:250–251], developed a classification scheme in terms of five different attributes:

1. *Time domain:* Calendar time or execution time,
2. *Category:* The total number of failures that can be experienced in infinite time is either *finite* or *infinite*,
3. *Type:* The distribution of the total number of failures experienced by time t (Poisson, Binomial,..., etc.,
4. *Class:* (for finite category only) functional form of the failure intensity in terms of time (exponential, Weibull, Gamma, ...etc.),
5. *Family:* (for infinite failure category only) functional form of the failure intensity in terms of the expected number of failures experienced (Geometric, Inverse linear, ... etc).

This classification scheme is given in Tables 2.1 and 2.2. Note that in the finite category table, class C indicates a distribution that does not have a common name. The same principle applies for the infinite category table, where types T indicate distributions that do not have common names.

2.3.4 Other Classification Schemes: There are many other classification schemes that can be found in the literature. Examples of other classifications include:

1. According to Applicable Phase

   Ramamoorthy and Bastani [78] classified software reliability models according to the phase where the software model applies. They classified software reliability models as follows:

Table 2.1. Finite Category Models [68:251]

Class	Type		
	Poisson	Binomial	Other Types
Exponential	Musa [63] Moranda [61] Schneidewind [80] Goel-Okumoto [33]	Jelinski-Moranda [38] Shooman[83]	Goel-Okumoto [31] Musa [65] Keiller-Littlewood [42]
Weibull		Schick-Wolverton [81] Wagoner [89]	
C1		Schick-Wolverton [82]	
Pareto		Littlewood [51]	
Gamma	Yamada-Ohba-Osaki [94]		

Table 2.2. Infinite Category Models [68:251]

Family	Type			
	T1	T2	T3	Poisson
Geometric	Moranda [61]			Musa-Okumoto [67]
Inverse Linear		Littlewood-Verrall [55]		
Inverse Polynomial 2nd degree			Littlewood-Verall [55]	
Power				Crow [17]

(a) Debugging Phase Models

(b) Validation Phase Models,

(c) Operational Phase Models,

(d) Maintenance Phase Models.

2. According to Nature of Debugging Strategy

   Bastani and Ramamoorthy [10] classified software reliability models according to the nature of debugging process as:

   (a) Reliability Growth Models,

   (b) Sampling Models,

   (c) Seeding Models.

From the above discussion, it is clear that most of the classifications are very similar.

2.4 Classical Software Reliability Models

As mentioned before, about 100 software reliability models have been developed to predict software reliability. In this section, a brief description of the most famous models will be given. The construction of many other models is based on the ideas illustrated by these models.

2.4.1 Jelinski-Moranda Model (JM): This model [38] is one of the early models that the reader finds in nearly all literature on software reliability modeling. It can be considered as the basis for most of the models that followed it. It was proposed by Jelinski and Moranda in 1972, taking into consideration the following assumptions:

1. A finite number of faults initially exists in the program,

2. Faults are independent, and each has the same probability of occurrence,

3. The debugging process is perfect, and takes place immediately when an error is detected,

4. The hazard rate decreases by an amount ϕ after discovery and correction of each fault (see Figure 2.3),

5. The time to next failure is proportional to the number of remaining faults in the program.

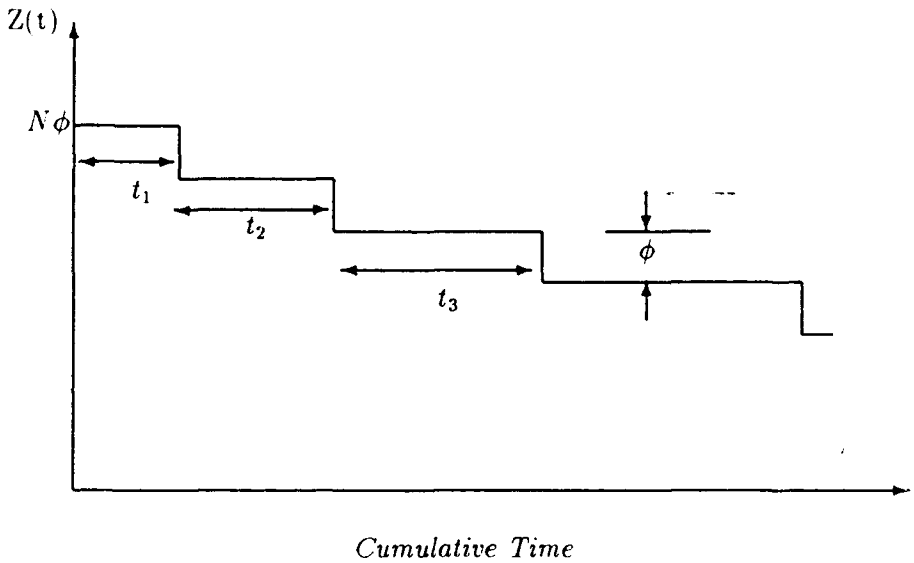

Figure 2.3. Z(t) Plot for JM model [38]

Using the above assumptions, the hazard rate $Z(t_i)$, between the $(i-1)^{\underline{st}}$ and the $i^{\underline{th}}$ failure is given by

$$Z(t_i) = \phi[N - (i-1)] \tag{2.1}$$

where

t_i = the time between the $(i-1)^{\underline{st}}$ and the $i^{\underline{th}}$ failure

ϕ = the drop in the hazard rate due to one fault removal

N = the initial number of faults in the program

Now, let $T_1, T_2, \cdots, T_n$ represent the time intervals between successive failures. Under the assumption of uniform failure rate (which implies that the times between failures are exponential with rate $Z(t_i)$), the probability density function of $T_i = (t_i - t_{i-1})$ is given by

$$f(T_i) = \phi[N - (i-1)]\exp\{-\phi[N - (i-1)]T_i\} \tag{2.2}$$

and the likelihood function is

$$L(T_1, T_2, \cdots, T_n) = \prod_{i=1}^{n} \phi[N - (i-1)]\exp\{-\phi[N - (i-1)]T_i\} \tag{2.3}$$

taking the partial derivative of $\ell n(L)$ with respect to N and ϕ and setting the resulting equations to 0, the MLE of N after n failures is obtained by solving the following equation and calling the result $\widehat{N}$

$$\frac{1}{N} + \frac{1}{N-1} + \cdots + \frac{1}{N-(n-1)} = \frac{n}{N - \frac{1}{T}\sum_{i=1}^{n}(i-1)T_i} \tag{2.4}$$

where $\quad T = \sum_{i=1}^{n} T_i$

The MLE of ϕ is found to be

$$\widehat{\phi} = \frac{n}{\widehat{N}T - \sum_{i=1}^{n}(i-1)T_i} \tag{2.5}$$

this concludes the second part of the model. The final part is simply "plugging in" to get required predictions. As an example, the MTTF after experiencing i failures, will be

$$MTTF(i) = \frac{1}{\widehat{\phi}(\widehat{N} - i)} \tag{2.6}$$

2.4.2 Bayesian Jelinski-Moranda Model (BJM): The motivation for the introduction of this model by Littlewood and Sofer [53] was that the JM model had produced too optimistic results with almost every set of data it was applied on. This was interpreted as a result of either the

unreality of the underlying assumptions of the model, or poor estimates of the parameters, due to the use of the maximum likelihood method. The BJM is a Bayesian approach to the JM model with slight modeling changes. It does not consider the failure rate to be an integer multiple of ϕ. The sequence of this model can be described as follows:

1. A "prior" distribution is assigned to the parameters λ (initial failure rate of the program) and ϕ (usually Gamma pdf),
2. Using the observed data $t_1, t_2, \cdots, t_{i-1}$, the distribution is modified to give the "posterior" distribution of λ and ϕ,
3. The model is then used to calculate the current reliability, the current failure rate, the mean time to failure, and the number of faults remaining in the program under test.

2.4.3 Goel-Okumoto Imperfect Debugging Model (GO): This model [31, 32], is another modification of the basic Jelinski-Moranda model. The motivation for the introduction of this model was the fact that in practice, the assumption of a perfect debugging process is proved to be unrealistic in most cases [60, 86, 87]. The modification was then made by introducing a probability α for imperfect debugging. The number of faults in the program at any time t, is treated as a Markovian process with the transition probability α. Times between transitions are taken to be exponentially distributed with rates depending on the number of faults remaining in the program. The hazard rate between the $(i-1)^{\underline{st}}$ and the $i^{\underline{th}}$ failure , $Z(t_i)$ is given as:

$$Z(t_i) = \lambda[N - \alpha(i-1)] \tag{2.7}$$

where

t_i = the time between the $(i-1)^{\underline{st}}$ and the $i^{\underline{th}}$ failure

λ = the failure rate per fault

Expressions are then derived for performance measures such as the distribution of time to a completely debugged program, distribution of the number of faults remaining in the program, and program reliability.

2.4.4 Littlewood-Verall Model (LV): Littlewood and Verall [55], took a different approach when they developed this model. Unlike most other models, they argued that software reliability should not be related to the number of faults in the program. In this model, it is possible that fixing a fault makes the program less reliable and even if an improvement took place, its magnitude is uncertain. The assumptions of the model are:

1. The times between failures $T_i's$ are assumed to be independent random variables (as in JM or BJM) with pdf

$$p(t_i \mid \lambda_i) = \lambda_i \exp(-\lambda_i t_i) \qquad (t_i > 0) \tag{2.8}$$

   i.e. the random variables $T_i's$ are exponential with parameter λ_i,

2. The $\lambda_i's$ form a sequence of random variables each has a gamma distribution with the parameters α and $\psi(i)$, i.e.,

$$f(\lambda_i \mid \alpha, \psi(i)) = \frac{[\psi(i)]^{\alpha} \lambda_i^{\alpha-1} \exp(-\psi(i)\lambda_i)}{\Gamma(\alpha)} \qquad \lambda_i > 0 \tag{2.9}$$

The function $\psi(i)$ is a linear function of i, reflecting the quality of the programmer and the difficulty of the programming task. A rapidly increasing function $\psi(i)$ reflects a good programmer, an easy task or both.

The user can choose the parametric family for $\psi(i)$. In [1], the parametric form of $\psi(i)$ was taken as

$$\psi(i, \beta) = \beta_1 + \beta_2 i \tag{2.10}$$

The MLE method is used to estimate the values of β_1, β_2 and α. Using the realized failure times $t_1, t_2, \cdots, t_{i-1}$, the reliability of the system is estimated to be [1:954]

$$\widehat{R}_i(t) = \left[\frac{\psi(i,\widehat{\beta})}{t + \psi(i,\widehat{\beta})}\right]^{\widehat{\alpha}} \tag{2.11}$$

where $\widehat{\alpha}$ and $\widehat{\beta}$ are the ML estimates of α, β. This model can also adjust the correction time to be a constant value greater than zero.

2.4.5 Keiler-Littlewood Model (KL): This model [41, 42] is similar to LV, except that reliability growth is induced via the shape parameter of the gamma distribution, i.e.,

$$f(\lambda_i \mid \beta, \psi(i)) = \frac{(\beta)^{\psi(i)} \lambda_i^{\psi(i)-1} \exp -\beta\lambda_i}{\Gamma(\psi(i))} \qquad \lambda_i > 0 \tag{2.12}$$

Here, reliability growth takes place when $\psi(i)$ is a decreasing function of i. Again $\psi(i)$ choice is under the user control. In [1:954], the parametric form of $\psi(i)$ was taken as

$$\psi(i,\alpha) = (\alpha_1 + \alpha_2 i)^{-1} \tag{2.13}$$

The MLE method is also used to estimate the values of α_1, α_2 and $\psi(i)$. Using realized failure times $t_1, t_2, \cdots, t_{i-1}$, the reliability of the system is estimated to be [1:954]

$$\widehat{R}_i(t) = \left[\frac{\widehat{\beta}}{t + \widehat{\beta}}\right]^{\psi(i,\widehat{\alpha})} \tag{2.14}$$

where $\widehat{\alpha}$ and $\widehat{\psi}(i)$ are the ML estimates of α, $\psi(i)$.

2.4.6 Musa Basic Execution Time Model: This model [63] is one of the most commonly used models in the software reliability field. What is interesting about this model is that it uses execution time (which is the real stress and meaningful time for software) instead of the calendar

time used by other models. On the other hand, this model can also transform execution time into calendar time, which is more convenient for software reliability managers and engineers. It is also possible to model the amount of limiting resources (testers, debuggers, and computer time) involved in the testing process.

The model starts with assuming the hazard rate function $Z(\tau)$ to be

$$Z(\tau) = fK(N_o - n(\tau)) \tag{2.15}$$

where

τ	=	amount of execution time used in testing
N_o	=	initial number of faults in the program
f	=	linear execution frequency (average instruction rate divided by the total number of instructions in the program)
K	=	error exposure ratio which relates the error exposure frequency to linear execution frequency
$n(\tau)$	=	number of faults detected by time τ

Assuming that the rate of failure occurrence is proportional to the rate of error correction, then

$$\frac{dn(\tau)}{d\tau} = BZ(\tau) \tag{2.16}$$

where B is the error reduction factor which represents the average ratio of the rate of reduction of errors to the rate of failure occurrence.

The above equation was further generalized by introducing a constant C to represent the ratio of rate of error detection in the testing phase to that in the operational phase, so Eq (2.16) can be expressed as

$$\frac{dn(\tau)}{d\tau} = BCZ(\tau) \tag{2.17}$$

combining Eqs (2.15) and (2.17), we get

$$\frac{dn(\tau)}{d\tau} = BCfK[N_o - n(\tau)] = BCfKN_o - BCfKn(\tau) \tag{2.18}$$

Knowing that $n = 0$ at $\tau = 0$, the solution of the above equation is found to be

$$n(\tau) = N_o[1 - \exp(-BCfK\tau)] \tag{2.19}$$

The $MTTF$ can then be calculated as

$$MTTF = 1/Z(\tau) = 1/[fKN_o \exp(-BCfK\tau)] \tag{2.20}$$

The reliability of the system at future execution time τ_1 will be

$$R(\tau_1) = \exp(-\int_0^{\tau_1} Z(x)dx) = \exp\left(\frac{-\tau_1}{MTTF}\right) \tag{2.21}$$

Other useful results that can be derived from this model include:

1. The number of faults to be detected to increase the time between failures from τ_1 to τ_2,
2. Additional CPU test time required to achieve the above goal,
3. The resource requirements needed for the same reason,
4. Utilization factor of the limiting resources.

2.4.7 Musa Logarithmic Poisson Execution Time Model: This is another execution time model developed by Musa and Okumoto [67] in 1984. It assumes an infinite number of faults in the program and the failure process is a NHPP. Further, the failure intensity λ is assumed to decrease exponentially with the number of removed faults, i.e.

$$\lambda(\tau) = \lambda_o exp(-\phi n(\tau)) \tag{2.22}$$

where λ_o is the initial failure intensity, and ϕ is called the failure intensity decay parameter.

Substituting for $\lambda(\tau)$, we get

$$\frac{dn(\tau)}{d\tau} = \lambda_o exp(-\phi n(\tau)) \tag{2.23}$$

Using the initial condition that $n(0) = 0$, the solution of the above differential equation is

$$n(\tau) = \frac{1}{\phi}\ell n(\lambda_o \phi \tau + 1) \tag{2.24}$$

It is clear from the above equation that the expected number of faults detected by execution time τ is a logarithmic function of τ, and hence the name of the model.

Using Eqs (2.22) and (2.24), the failure intensity at execution time τ is

$$\lambda(\tau) = \lambda_o exp[-\phi n(\tau)] = \frac{\lambda_o}{\lambda_o \phi \tau + 1} \tag{2.25}$$

The reliability function at execution time τ_o is

$$R(\tau \mid \tau_o) = \left(\frac{\lambda_o \phi \tau + 1}{\lambda_o \phi (\tau + \tau_o) + 1}\right)^{1/\phi} \quad , \tau \geq 0 \tag{2.26}$$

Other reliability measures can be obtained using the general theories of the NHPP models, as described in Musa and Okumoto [67].

III. SOFTWARE RELIABILITY MODEL SELECTION

Software Reliability models differ considerably in the ways they transform model assumptions into a detailed mathematical structure. The problem of proper model selection can be summarized briefly as follows [52:149]: The available set of data to the user will be a sequence of times between successive failures $t_1, t_2, ..., t_{i-1}$. These observed times are the realization of random variables $T_1, T_2, ..., T_{i-1}$. A software reliability model is then applied to use the observed set of data $t_1, t_2, ..., t_{i-1}$, to predict the future unobserved $T_i, T_{i+1}, \cdots$. The problem to be solved by a software reliability model is then a prediction problem. It involves the future via the unobserved random variable T_i. Using the goodness of fit will not solve the problem. The reason, as pointed out in [1:951] is that models are usually too complicated for a traditional "goodness-of-fit" approach to be used. In the literature, there are many suggestions for solving the problem of proper model selection. Some of these suggestions will be given in this chapter.

3.1 Model Selection Using Analysis of Predictive Quality

The idea behind this method is to do some analysis about the predictive quality measures (such as accuracy, bias, trend, noise, $\cdots$ etc.) for candidate software reliability models, with respect to the set of data in hand. Detailed examples of these analyses can be found in [1] , [41], [42], and [52]. A brief summary of such methods will be given in the following discussion.

Given that the observed values $t_1, t_2, \cdots, t_{i-1}$ of the random variables $T_1, T_2, \cdots, T_{i-1}$, the manager of the software product will be interested in having a good prediction for the future T_i. This future T_i, can then be transformed into the current reliability of the software product as follows:

$$R_i(t) = 1 - P(T_i < t) = 1 - F_i(t) \tag{3.1}$$

The user will never know, even at a later stage, the exact value of $F_i(t)$. However, by using a software reliability model the estimated value $\widehat{F}_i(t)$ can be calculated. By realizing the actual value of T_i at the time of next failure, the user will have a pair $\{\widehat{F}_i(t), t_i\}$. After having enough pairs of such data, the user can tell whether there is any evidence to suggest that the $t_i's$ are realizations of random variables from $F_i(t)'s$. If such evidence were not found, it would suggest that there are significant differences between $\widehat{F}_i(t)$ and $F_i(t)$; hence; the predictions are not in accord with actual behavior. Littlewood [52:166] made the following sequence of transformations:

$$u_i = \widehat{F}_i(t_i) \tag{3.2}$$

Each is a probability integral transform of the observed t_i, using the previously calculated predictor $\widehat{F}_i$, based upon $t_1, t_2, \cdots, t_{i-1}$. According to Dawid [22] and Rosenblatt [79], if each $\widehat{F}_i$ were identical to the true F_i, then u_i would be realizations of independent uniform $U(0,1)$ random variables.

3.1.1 the u-Plot: The main idea behind this method is that since $\{u_i's\}$ are supposed to have a uniform distribution, then the quality of the software reliability model can be measured by whether the sequence of $u_i's$ looks like a random sample from $U(0,1)$. Knowing that the cumulative distribution function (cdf) of $U(0,1)$ is a line of unit slope through the origin, the u-Plot can be used to judge the prediction quality of a software reliability model as foliows:

1. place the u_i values on the horizontal axis,
2. At each point, increase the step function by $1/(n+1)$,
3. Compare the resulting plot with the line of unit slope,
4. The deviation from this line, which can be measured using the Kolmogorov distance (maximum absolute vertical difference) will be used to measure the quality of the software reliability model.

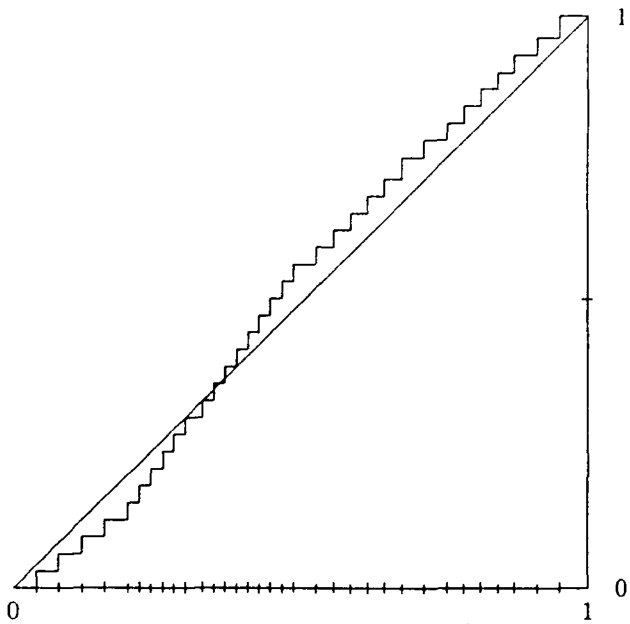

Figure 3.1. Drawing the u-Plot (Littlewood [52:168])

Another quality metric that can be deduced using the u-Plot is the *bias* of the model. If all of the $u_i's$ are above the line of unit slope, then this means that the model predictions tend to be too optimistic and vise versa if all of the $u_i's$ are under the line of unit slope.

3.1.2 The y-Plot and Scatter Plot of $u_i's$: One problem with the u-Plot is that it measures only one type of departure of predictors from reality. For example if a prediction system applied for a set of data shows optimism in the early predictions, and pessimism in the later predictions, then a small Kolmogorov distance will be observed. It seems necessary then to examine the $u_i's$ for *trend*. To do that, the y-Plot method makes the following transformation

$$x_i = -\ell n(1 - u_i) \qquad (3.3)$$

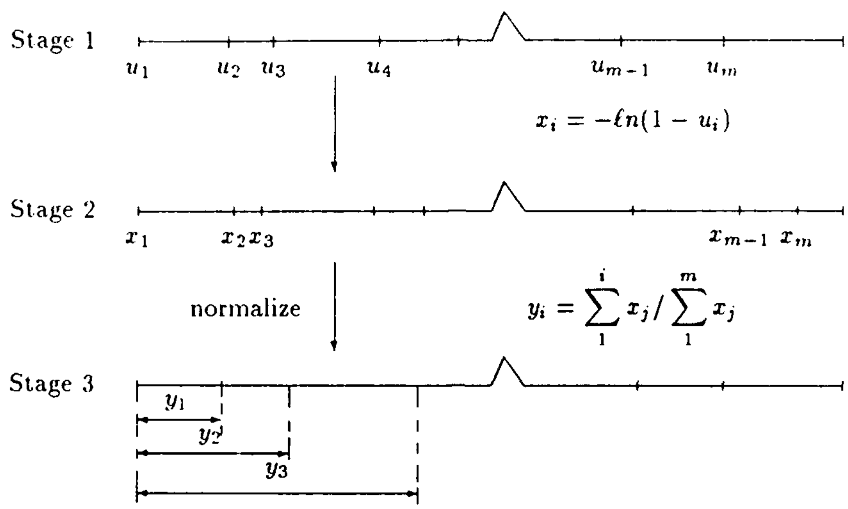

Figure 3.2. Transformations to Obtain the y-Plot (Abdel-Ghaly [1:958])

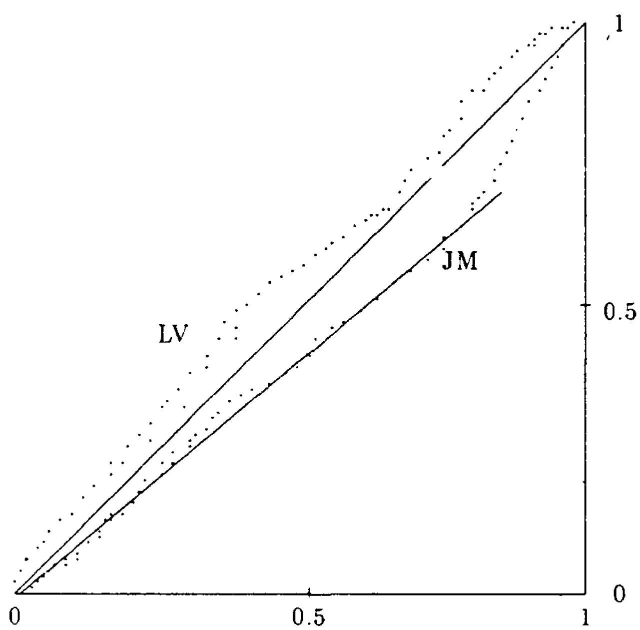

Figure 3.3. LV and JM y-Plots for Data of Table 1 in Littlewood [52:172]

If the $\{u_i's\}$ are a true realization of *iid* random variables with $U(0,1)$ distribution, then the $\{x_i's\}$ will be a realization of *iid* unit exponential random variables. Thus, if the $\{x_i's\}$ are plotted on the horizontal axis, then they should look like a realization of a homogeneous Poisson process. The alternative hypothesis (that there is trend in the $u_i's$) will show itself as a non-constant rate for this process. If the values of $\{x_i's\}$ are normalized onto (0,1) and their values are plotted, then a plot that shows the trend of the predictions of the model will result (see Figure 3.3). Departure from the line of unit slope will indicate that the prediction system is not capturing the "trend" in the failure data (i.e. reliability growth) [41, 42]. Figure (3.3) shows the application of the y-Plot procedure to the Littlewood-Verall (LV) and Jelinski-Moranda (JM) predictions of the table 1 data in [52:144]. The Kolmogorov distances are found to be 0.120 for (JM) and 0.110 for (LV), both of them are not significant at the 10% level [52:170]. A close look at the JM y-Plot shows that it is very close to linearity at early stages (until about i=90: see the fitting line).

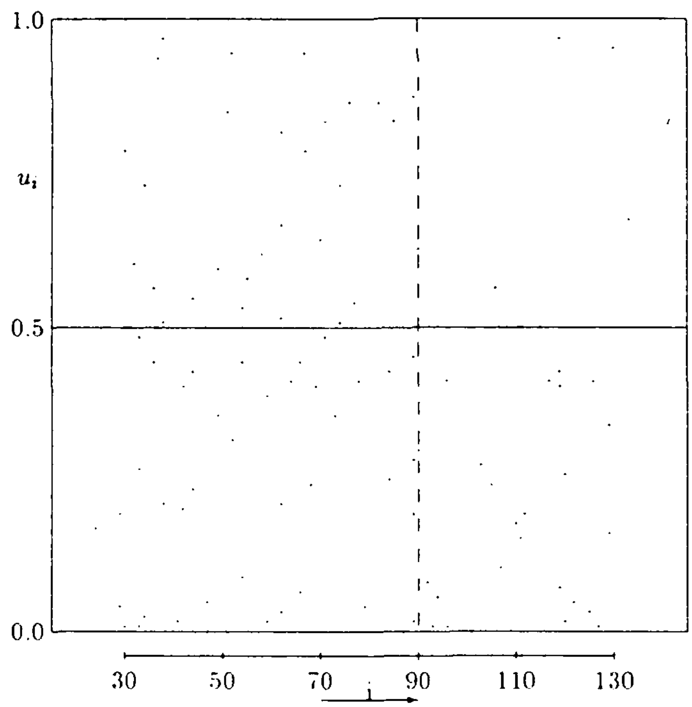

Figure 3.4. Scatter Plot of u_i for JM Model Using Data of Table 1 in [52:173]

In the scatter plot, the values of $\{u_i's\}$ are plotted as in Figure (3.4) and the trend can be judged by the number, and location of points in each stage. As an example, it is clear from the plot that the prediction system seems to be too optimistic after $i = 90$. This is mainly because the number of points with $\{u_i's\}$ less than 0.5 after $i = 90$ is higher than those with u_i greater than 0.5.

3.1.3 Measures of Noise: Both the u-Plot and the y-Plot can be considered as a measure of bias. In the sequel some tools will be used to analyze the variability of the prediction. These tools are, in fact, some quite crude measures of variability, due to the problem of unavailability of the true sequence of $F_i(t)$ [52:175].

3.1.3.1 Braun Statistic: This is a measure of predictions noise. It is defined as [13]:

$$B = \frac{\sum_i (t_i - \hat{E}(T_i))^2}{\sum_i (t_i - t)^2} \cdot \frac{n-1}{n-2} \tag{3.4}$$

where

$\hat{E}(T_i)$	=	the estimated mean of T_i
n	=	number of the terms in the sum

A small value of B indicates better smoothing of software reliability model predictions.

3.1.3.2 Median Variability: This is another measure of noise, which is defined as

$$MV = \sum_i \frac{m_i - m_{i-1}}{m_{i-1}} \tag{3.5}$$

where m_i is the predicted median of T_i. Between different prediction systems, this measure can indicate which model has higher variability in its predictions, or in other words, how noisy the model predictions are. Again, a small value of MV indicates better smoothing of the predictions.

3.1.3.3 Rate Variability: The rate variability (RV) is defined as

$$RV = \sum_{i} \frac{r_i - r_{i-1}}{r_{i-1}} \qquad (3.6)$$

where r_i, the rate of occurrence of failures ($ROCOF$), is calculated immediately after a fix. This is also another measure of noise. It is important to note that for both MV and RV, the comparison between two prediction systems must be for the same set of data.

3.1.4 Prequential Likelihood: Most of the work concerning the Prequential Likelihood was done by Dawid [21, 22]. The Prequential Likelihood (PL) for one-step ahead prediction of $T_{j+1}, T_{j+2}, \cdots, T_{j+n}$ is defined as follows[52:177]:

$$PL_n = \prod_{i=j+1}^{j+n} \widehat{f}_i(t) \qquad (3.7)$$

where $\widehat{f}_i(t)$ is the pdf of the predictive distribution $\widehat{F}_i(t)$, based on the realized values $t_1, t_2, \cdots, t_{i-1}$.

The PL value can be used to determine the accuracy of a software reliability model [57:174]. This value is usually very small, therefore its logarithmic value (which is always negative) is used for comparison. The more negative it is, the more inaccurate the software reliability model predictions. This measure can be considered as a general procedure for choosing the best prediction system for a given set of data [14:461], so in case of having a tie between two models with respect to all other quality measures, the one with higher accuracy will be preferable [57:174].

A comparison of two prediction systems A and B, over a range of predictions $T_{j+1}, T_{j+2}, \cdots, T_{j+n}$ can be made via their prequential likelihood ratio $(PLR)_{n,A,B}$

$$(PLR)_{n,A,B} = \frac{\prod_{i=j+1}^{j+n} \widehat{f}_i^A(t_i)}{\prod_{i=j+1}^{j+n} \widehat{f}_i^B(t_i)} \qquad (3.8)$$

Dawid [22:283–284] stated that if $PLR_{n,A,B} \to \infty$ as $n \to \infty$, then prediction system A will be considered better than B. Detailed discussion about these measures can be found in [1],[21], [22], and [37].

Model	u-plot K-S dist. (sig. level)	y-plot K-S dist. (sig.level)	Braun Statistic (rank)	Median Variability (rank)	Rate of Variability (rank)	$-log_e$(PL) (rank)
Jelinski-Moranda JM [38]	.121 (NS)	.115 (NS)	1.11 (8)	4.23 (6)	3.81 (8)	466.22 (6)
Bayesian JM BJM [53]	.110 (NS)	.077 (NS)	1.04 (5)		3.27 (7)	466.64 (7)
Littlewood L [51]	.123 (NS)	.091 (NS)	1.07 (7)	5.27 (7)	4.66 (9)	465.49 (2)
Bayesian Littlewood BL [2]	.138 (NS)	.068 (NS)	.96 (1)		2.82 (6)	465.18 (1)
Littlewood-Verall LV [55]	.167 (10%)	.051 (NS)	.97 (3)	2.33 (3)	2.18 (4)	465.52 (3)
Keiller-Littlewood KL [41, 42]	.170 (10%)	.051 (NS)	.96 (1)	2.33 (3)	2.17 (3)	465.81 (4)
Duane D [27, 18]	.209 (2%)	.052 (NS)	1.04 (5)	1.96 (2)	1.84 (2)	467.78 (9)
Goel-Okumoto GO [31]	.271 (1%)	.085 (NS)	1.21 (10)	1.06 (1)	1.03 (1)	473.87 (10)
Littlewood NHPP LNHPP[1]	.169 (10%)	.082 (NS)	.97 (3)	3.05 (5)	2.76 (5)	465.85 (5)
Weibull W [2]	.100 (NS)	.111 (NS)	1.17 (9)	6.16 (8)	5.48 (10)	466.89 (8)

Table 3.1. Littlewood's Analysis of Data of Table 2 in [52:189]

Table (3.1) is a summary of applying the above-mentioned prediction quality measures. From this table, it is clear that different methods of model selection result in different models being chosen. This problem, and the fact that some of the above methods are rather subjective as to which model is better than others [45:184], force the user to try different measures and take the average, or to judge the results from his own perspective.

3.2 Model Selection Based on a Linear Combination of Models

The idea behind this method is that, rather than predicting software reliability by using only one model, a *meta predictor* could be formed to take a linear combination of two (or more) predictions, with weights chosen in some optimal way.

The heuristic algorithm for this method is as follows [57:173]

1. Identify the candidate set of software reliability models to be used.

2. Select the models that tend to cancel out in their biased predictions (if any).

3. Selected models are equally weighted to form a linear combination model.

4. The arithmetic average (*mean value*) of the predictions, resulting from the selected models or their middle prediction value (*median value*), is used as the prediction of the linear combination of models.

The authors used the Goel-Okumoto NHPP Model (GO) [33], the Musa-Okumoto Logarithmic Model (MO) [67], and the Littlewood-Verall Model (LV) [55] to form an Equally-Weighted Linear Combination (ELC) Model. The arithmetic average of each selected models's predictions was taken as the ELC Model predictions as follows:

$$ELC = \frac{1}{3}GO + \frac{1}{3}MO + \frac{1}{3}LV \tag{3.9}$$

The main drawback of this method is that it is not the best in all cases. Also it mixes models of different assumptions for the failure process.

3.3 Model Selection Based on Check of Limits

In this approach, a check of the limit conditions for the model to be used is performed before using it, so as to avoid the meaningless consumption of time [93:481]. The check of the limit conditions confirms the convergence of the model. As an example of this technique, H. Xizi [93:483] derived the limit condition for the Musa Execution Time Model [63] as:

$$\frac{\sum_{i=1}^{m}(i-1)t_i}{\sum_{i=1}^{m}t_i} < \frac{m-1}{2} \tag{3.10}$$

where

$m \quad = \quad n_c/B$

$n_c \quad =$ number of errors corrected

$B \quad =$ error reduction factor

$t_i \quad =$ the time between the $(i-1)^{\underline{st}}$ and the $i^{\underline{th}}$ failures, $i = 1, 2, \cdots, m$

Littlewood [54:145] presented the convergence condition for the Jelinski-Moranda model (JM) [38] as:

$$\frac{\sum_{i=1}^{n}(i-1)t_i}{\sum_{i=1}^{n}(i-1)} < \frac{\sum_{i=1}^{n} t_i}{n} \tag{3.11}$$

where

$n \quad =$ observed number of failures

$t_i \quad =$ the time between the $(i-1)^{\underline{st}}$ and the $i^{\underline{th}}$ failures $i = 1, 2, \cdots, n$

The problem with this method is that the limit conditions for most of the models are similar, which makes the user return to the the same problem of proper model selection.

3.4 Model Selection Using Akaike Information Criterion (AIC)

The (AIC) was developed by Akaiki [3] in 1974. It resulted from relating the entropy principle from statistical mechanics to information theory. In a subsequent work [4], Akaiki proved that the (AIC) can be used to select the model that best predicts the stochastic behavior of the system.

The use of the Akaike Information Criterion for software reliability model selection was first suggested by Khoshgoftaar [43]. In a subsequent work [44], he showed that this technique, when applied with some extensive simulation work, proved the feasibility of using the (AIC) for software reliability model selection.

Entropy in statistical mechanics is the measure of the probability of having a system of particles in state A. It increases with the system changing to its most probable state B. If A=B, entropy will be maximum and can be taken as the measure of how close the estimator is to its real distribution. For two probability density functions $f(z)$ and $g(z)$, the entropy of distribution $f(z)$ with respect to distribution $g(z)$, denoted as $B(f(z), g(z))$ is expressed as [45:186]

$$B(f(z), g(z)) = -\int f(z)\ell n\left(\frac{f(z)}{g(z)}\right)dz \tag{3.12}$$

$$= -E_f\left[\ell n\left(\frac{f(z)}{g(z)}\right)\right] + C \tag{3.13}$$

$$= -I(f(z); g(z)), \tag{3.14}$$

where

E_f = the expectation with respect to distribution $f(z)$

$I(f(z); g(z))$ = the expected log-likelihood ratio, or the negative of entropy

C = a constant value

The software reliability model that minimizes

$$-2\ell n\left(\frac{f(z)}{g(z)}\right) + 2C \tag{3.15}$$

is the model that should be chosen Khoshgoftaar and Woodcock [45:187] stated that the AIC can be written as the following expression:

AIC = -2(log likelihood function at its maximum likelihood estimators) +2 (number of parameters fitted when maximizing the likelihood function)

The best model for a given data set using this method is that with the lowest AIC value.

3.5 Recalibrating Software Reliability Models

This method was introduced by Brocklehurst and Others [14], and can be summarized as follows:

The relation between true distribution $F_i(t)$ of the random variable T_i, and the predicted one $\widehat{F}_i(t)$ can be represented through a relation function G_i as

$$F_i(t) = G_i\left[\widehat{F}_i(t)\right] \tag{3.16}$$

It is further assumed that G_i is only slowly changing function with i. Since G_i is not known, it will be approximated with an estimate G_i^* which will lead to a new prediction

$$\widehat{F}_i^*(t) = G_i^*\left[\widehat{F}_i(t)\right] \tag{3.17}$$

This, in fact, recalibrates the raw model output $\widehat{F}_i(t)$ in the light of the accuracy of past predictions for the data set under study. The authors based their estimate of G_i^* on the u-Plot, calculated from predictions which have be made prior to T_i. They made two choices for the shape of $G_i^*(t)$. The first choice is a simple shape in the form of a u-Plot with steps joined up to form a polygon, while the second one is a more complicated one in the form of a spline.

The sequence of this method can be summarized as follows [14:461]:

1. Use the y-Plot to make sure that the error in previous predictions is approximately stationary,
2. Construct the polygon G_i^* by joining up the steps of the u-Plot formed by predictions before T_i, see Figure (3.5),
3. Get the raw prediction $\widehat{F}_i(t)$ by using the basic prediction system,
4. Recalibrate the raw predictions using Eq (3.17).

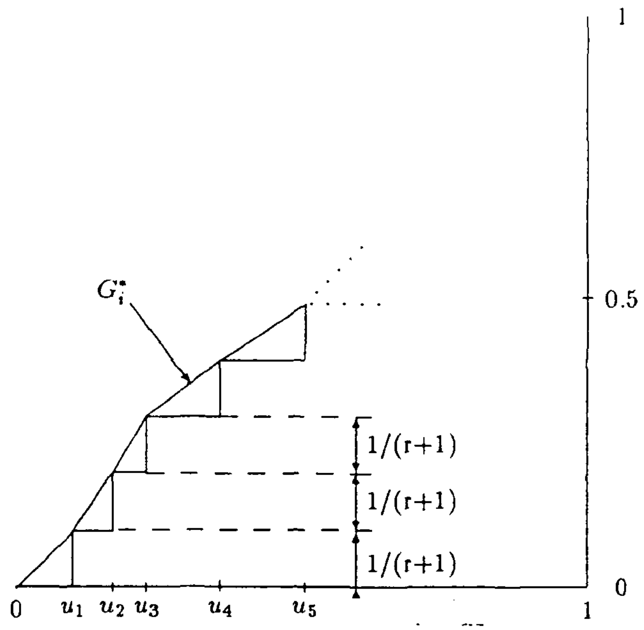

Figure 3.5. Drawing the Step Recalibrating Function G_i^* (Brocklehurst [14:461])

The main problem with this technique, as stated by the authors, is that the failure behavior of software cannot be guaranteed to be stationary in all cases. Also, recalibrated predictions can sometimes be worse than raw predictions.

IV. MATHEMATICAL BACKGROUND

In this chapter, several mathematical concepts of the reliability theory will be presented. These concepts are essential for understanding the process of software reliability modeling. The most important distributions in reliability theory and the relations between these distributions are also given in this chapter. The chapter concludes with a derivation of the test statistic and its moments.

Throughout this discussion, the random variable T will be used to represent the time to next failure, and t will represent the realization of T.

4.1 Fundamental Reliability Equations

There are some principles in hardware reliability theory that are also applicable in software reliability theory.

The first common principle is that the *reliability of a system (program)* at time t, $R(t)$ is defined as the probability that there will be no failures by t, i. e.

$$R(t) = P(T > t) = 1 - P(T \leq t) = 1 - F(t), \tag{4.1}$$

where $F(t)$ is the probability of failure before time t.

Also, the *failure rate* is defined as the probability that a failure per unit time occurs in the interval between two times (t_1, t_2), given that the system (*program*) has survived until time t_1:

$$\frac{P(t_1 \leq T < t_2 \mid T > t_1)}{(t_2 - t_1)} = \frac{P(t_1 \leq T < t_2)}{(t_2 - t_1)P(T > t_1)} \tag{4.2}$$

$$= \frac{F(t_2) - F(t_1)}{(t_2 - t_1)R(t_1)} \tag{4.3}$$

The *hazard rate* $Z(t)$ for a component (*software program*) is defined as the instantaneous failure rate at time t. Therefore, it can be expressed as the limiting condition of Eq (4.3) as the interval $(t_2 - t_1)$ approaches 0, so

$$Z(t) = \lim_{\Delta t \to 0} \frac{F(t+\Delta t) - F(t)}{\Delta t R(t)} = \frac{f(t)}{R(t)} = \frac{f(t)}{1 - F(t)} \tag{4.4}$$

where $f(t)$ is the failure density or the first derivative of $F(t)$.

The relation between $Z(t)$ and $R(t)$ rate can be derived by integrating the two sides of Eq (4.4), i.e.

$$R(t) = \exp[-\int_0^t Z(x)dx] \tag{4.5}$$

Using Eqs (4.1), (4.4), and (4.5), we get

$$f(t) = Z(t)\exp[-\int_0^t Z(x)dx] \tag{4.6}$$

The mean time to failure (MTTF), is defined as the expected time to the next failure, and can be expressed as

$$MTTF = E(T) = \int_0^\infty tf(t)dt \tag{4.7}$$

It can also be computed by the following relation [40:10]

$$MTTF = \int_0^\infty R(t)dt \tag{4.8}$$

If $N(t)$ is the number of failures experienced by time t, then the mean value of $N(t)$ is $\mu(t) = \mathrm{E[N(t)]}$, which is an increasing function of t, and its derivative is the failure intensity $\lambda(t) = d\mu(t)/dt$.

4.2 Some Important Reliability Distributions

In this section, some of the important distributions used in reliability studies will be presented, along with the relations between them. There are also some theories related to these distributions, which are needed for the derivation of the test criteria.

4.2.1 The Poisson Distribution: The series

$$1 + m + \frac{m^2}{2!} + \frac{m^3}{3!} + \cdots = \sum_{x=0}^{\infty} \frac{m^x}{x!} \tag{4.9}$$

converges for all values of m to e^m. Consider the function $f(x)$ defined by

$$f(x) = \begin{cases} \frac{m^x e^{-m}}{x!} & x = 0, 1, 2, \ldots \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

where $m > 0$. This makes $f(x) \geq 0$ and

$$\begin{aligned} \sum_{x=0}^{\infty} f(x) &= \sum_{x=0}^{\infty} \frac{m^x e^{-m}}{x!} \\ &= e^{-m} \sum_{x=0}^{\infty} \frac{m^x}{x!} = e^{-m} e^m = 1 \end{aligned} \tag{4.11}$$

thus, $f(x)$ satisfies the conditions of being a pdf of a discrete type random variable. A random variable X which has a pdf of that form of $f(x)$ is said to have a *Poisson* distribution, and any such $f(x)$ is called a *Poisson pdf.*

4.2.2 The Gamma Distribution: For any positive number α, it is known that the integral

$$\int_0^{\infty} y^{\alpha-1} e^{-y}\, dy$$

exists for $\alpha > 0$ [23:235], and that the value of that integral is a positive number called the Gamma function of α, where

$$\Gamma(\alpha) = \int_0^\infty y^{\alpha-1} e^{-y} dy \tag{4.12}$$

If $\alpha = 1$, it is clear that

$$\Gamma(1) = \int_0^\infty e^{-y} dy = 1 \tag{4.13}$$

If $\alpha > 1$, then an integration by parts shows that

$$\Gamma(\alpha) = (\alpha - 1) \int_0^\infty y^{\alpha-2} e^{-y} dy = (\alpha - 1) \cdot \Gamma(\alpha - 1) \tag{4.14}$$

Accordingly, if α is a positive integer greater than 1, then

$$\Gamma(\alpha) = (\alpha - 1)(\alpha - 2)...(3)(2)(1)\Gamma(1) = (\alpha - 1)! \tag{4.15}$$

In the integral that defines $\Gamma(\alpha)$, let us introduce a new variable x by writing $y = x/\beta$, where $\beta > 0$, then

$$\Gamma(\alpha) = \int_0^\infty \left(\frac{x}{\beta}\right)^{\alpha-1} e^{-x/\beta} \left(\frac{1}{\beta}\right) dx, \tag{4.16}$$

Dividing by $\Gamma(\alpha)$, we get

$$1 = \int_0^\infty \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-x/\beta} dx \tag{4.17}$$

Since $\alpha > 0$, $\beta > 0$ and $\Gamma(\alpha) > 0$, it is clear that

$$f(x;\alpha,\beta) = \begin{cases} \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-x/\beta} & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.18}$$

is a pdf of a random variable of the continuous type. A random variable X that has a pdf of that form is said to have a *Gamma distribution*($X \sim \Gamma(\alpha, \beta)$).

From the properties of the pdf, it is known that

$$\int_0^\infty \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-x/\beta} dx = 1 \tag{4.19}$$

which can be rewritten as

$$\int_0^\infty x^{\alpha-1} e^{-x/\beta} dx = \beta^\alpha \Gamma(\alpha) \tag{4.20}$$

The mean and variance of a random variable X having a Gamma distribution are

$$E(X) = \mu = \alpha\beta \qquad V(X) = \sigma^2 = \alpha\beta^2$$

4.2.3 The Chi-Squared Distribution: If v is any positive integer (usually referred to as degree of freedom), then a random variable X is said to have a ***Chi-Squared distribution*** with parameter v ($X \sim \chi^2(v)$) if the pdf of X is the Gamma density defined in (4.18) with $\alpha = v/2$ and $\beta = 2$. Thus, the pdf of a chi-squared random variable X is

$$f(x; v) = \begin{cases} \frac{1}{\Gamma(v/2)2^{v/2}} x^{(v/2)-1} e^{-x/2} & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.21}$$

The mean and variance of a random variable X having a Chi-Squared distribution are

$$E(X) = \mu = v \qquad V(X) = \sigma^2 = 2v$$

4.2.4 The One-Parameter Exponential Distribution: The one-parameter exponential distribution is another special case of the general Gamma distribution defined in (4.18) in which

$\alpha = 1$ and β has been replaced by σ i.e. X is said to have a *one-parameter exponential distribution* $(X \sim EXP(\sigma))$ if the pdf of X is

$$f(x;\sigma) = \begin{cases} \frac{1}{\sigma}e^{-\frac{x}{\sigma}} & x \geq 0, \sigma > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.22}$$

The mean and variance of a random variable X having a one-parameter exponential distribution are

$$E(X) = \mu = \sigma \qquad \textit{and} \qquad V(X) = \sigma^2$$

and the cumulative density function (cdf) is

$$F(x;\sigma) = 1 - e^{-(x/\sigma)} \tag{4.23}$$

4.2.5 TheTwo-Parameter Exponential Distribution: A random variable X is said to have a two-parameter exponential distribution with parameters θ and σ $(X \sim EXP(\sigma;\theta))$ if it has a pdf

$$f(x;\theta,\sigma) = \begin{cases} \frac{1}{\sigma}e^{-\frac{x-\theta}{\sigma}} & x \geq \theta, \sigma > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

where θ is the *"threshold"* or *"guarantee time"* and σ_i is usually called *"the mean time between failures"*.

From the definition of the pdf, one can write

$$\int_0^x \frac{1}{\sigma}e^{-(\frac{x-\theta}{\sigma})}dx = \int_\theta^x \frac{1}{\sigma}e^{-(\frac{x-\theta}{\sigma})}dx = 1 \tag{4.25}$$

multiplying both sides by σ yields

$$\int_0^x \epsilon^{-(\frac{x-\theta}{\sigma})}dx = \int_\theta^x \epsilon^{-(\frac{x-\theta}{\sigma})}dx = \sigma \tag{4.26}$$

The cdf of the two-parameter exponential distribution is

$$\begin{aligned}
F(x;\sigma,\theta) &= \int_{-\infty}^x f(x)dx & &= \int_\theta^x \frac{1}{\sigma}\epsilon^{-(\frac{x-\theta}{\sigma})}dx \\
&= \int_\theta^x \frac{1}{\sigma}\epsilon^{-(\frac{x}{\sigma})}\epsilon^{(\frac{\theta}{\sigma})}dx & &= \epsilon^{(\frac{\theta}{\sigma})}\int_\theta^x \frac{1}{\sigma}\epsilon^{-(\frac{x}{\sigma})}dx \\
&= -\epsilon^{(\frac{\theta}{\sigma})}[\epsilon^{-(\frac{x}{\sigma})}]\,|_\theta^x & &= -e^{(\frac{\theta}{\sigma})}\left[e^{-(\frac{x}{\sigma})} - e^{-(\frac{\theta}{\sigma})}\right] \\
&= 1 - \epsilon^{-[\frac{(x-\theta)}{\sigma}]}
\end{aligned} \tag{4.27}$$

4.3 The Moment Generating Function

The mean and the variance of a random variable X are special cases of what is more generally referred to as the moments of the random variable. The mean $E(x)$ is *the first moment around the origin*, and the variance σ^2 is *the second moment about the mean*. Higher moments are also useful for characterizing other distribution aspects (e.g. the *skewness* of the pdf can be measured in terms of its third moment about the mean).

The classic method for obtaining the $r^{\underline{th}}$ moment of a random variable X is to evaluate the integration $\int_{-\infty}^{\infty} x^r f(x)dx$. Integrals of that form are not always easy to evaluate. An alternative, and easy, method is to evaluate the *moment generating function* $M_x(t)$ which can be defined as

Definition IV.1 *For all real values of t for which the expected value exists, the moment generating function (m.g.f.) for a random variable X with a pdf f(x), denoted $M_x(t)$ is*

$$M_x(t) = E(\epsilon^{tx}) = \int_{-\infty}^{\infty} e^{tx} f(x)dx \tag{4.28}$$

One of the most important properties of $M_x(t)$ is that its $r^{\underline{th}}$ derivative evaluated at zero ($M_x^r(t)|_0$) is equal to $E(X^r)$ [23:163]. If $M_x(t)$ can be found, then $M_x^r(t)$ will be easier to evaluate than $\int_{-\infty}^{\infty} x^r f(x)dx$.

Applying the above definition for the Gamma distribution defined in (4.18), we get

$$M_x(t) = \int_0^{\infty} e^{tx}\frac{1}{\beta^\alpha\Gamma(\alpha)}x^{\alpha-1}e^{-x/\beta}dx = \frac{1}{\beta^\alpha\Gamma(\alpha)}\int_0^{\infty} x^{\alpha-1}e^{-x(\frac{1}{\beta}-t)}dx \tag{4.29}$$

using Eq (4.20), this is equivalent to

$$\frac{1}{\beta^\alpha\Gamma(\alpha)}\cdot\frac{\Gamma(\alpha)}{(\frac{1}{\beta}-t)^\alpha} = \frac{1}{(1-\beta t)^\alpha} = (1-\beta t)^{-\alpha}$$

i.e., for the Gamma distribution

$$M_x(t) = (1-\beta t)^{-\alpha} \tag{4.30}$$

For the Chi-squared distribution $M_x(t)$ is

$$M_x(t) = (1-2t)^{-v/2} \tag{4.31}$$

and for the One-Parameter Exponential distribution

$$M_x(t) = (1-\sigma t)^{-1} \tag{4.32}$$

4.4 Some Important Relations

In the following, the relations between the different distributions will be explored.

Theorem IV.1 *If X has a two-parameter exponential pdf, $f(x;\theta,\sigma)$ defined in Eq (4.24), then $(X - \theta)$ has a one-parameter pdf $f(x;\sigma)$ defined in Eq (4.22).*

Proof

Let $Y = X - \theta$. From the hypothesis, we have $X \sim EXP(\theta, \sigma)$, where $EXP(\theta, \sigma)$ is exponential distribution described in (4.24). The following lemma [9:245] is required to prove this theorem.

Lemma IV.1 *Given the pdf of X, then the pdf of Y where g(X) = Y is*

$$g(y) = f(g^{-1}(y)) \left| \frac{d}{dy} g^{-1}(y) \right| \tag{4.33}$$

From the assumption, it follows that $X = g^{-1}(y) = Y + \theta$ and

$$\left| \frac{d}{dy} g^{-1}(y) \right| = 1 \tag{4.34}$$

Therefore, the pdf of Y is

$$\begin{aligned} g(y) &= f(x; \theta, \sigma) \left| \frac{d}{dy} g^{-1}(y) \right| = \frac{1}{\sigma} e^{\frac{-(x-\theta)}{\sigma}} \cdot |1| \\ &= \frac{1}{\sigma} e^{\frac{-(y+\theta-\theta)}{\sigma}} \\ &= \frac{1}{\sigma} e^{\frac{-y}{\sigma}} \end{aligned} \tag{4.35}$$

i.e., the pdf of Y can be written as

$$g(y) = \begin{cases} \frac{1}{\sigma} e^{-\frac{y}{\sigma}} & y \geq 0, \sigma > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.36}$$

which is the pdf of a one-parameter exponential distribution.

Lemma IV.2 *If $M_x(t)$ is the moment generating function of X, then*

$$M_{ax+b}(t) = e^{tb} \cdot M_x(at)$$

Proof

$$\begin{aligned} M_{ax+b}(t) &= E(e^{t(ax+b)}) &= E(e^{tax} \cdot e^{tb}) \\ &= e^{tb}E(e^{tax}) &= e^{tb} \cdot M_x(at) \end{aligned}$$

Lemma IV.3 *The moment generating function (m.g.f.) of X, where X has a two-parameter exponential pdf is*

$$M_x(t) = e^{t\theta}(1 - \sigma t)^{-1}$$

Proof

Let $Y = X - \theta$, then Y will have a one-parameter exponential distribution whose moment generating function (m.g.f.) is

$$M_y(t) = (1 - \sigma t)^{-1} \tag{4.37}$$

since $X = Y + \theta$, then using lemma(IV.2) and Eq (4.32), we get

$$M_x(t) = e^{t\theta}(1 - \sigma t)^{-1} \tag{4.38}$$

Theorem IV.2 *Suppose that $X_1, X_2, \cdots, X_n$ are independent, and X_i has m.g.f. $M_{X_i}(t)$, where $i = 1, 2, \cdots, n$, then the m.g.f. of $a_1X_1 + a_2X_2 + \cdots + a_nX_n$ is*

$$M_{a_1x_1+a_2x_2+\cdots+a_nx_n}(t) = M_{x1}(a_1t)M_{x2}(a_2t)\cdots M_{xn}(a_nt)$$

Proof

The proof of this theorem is given in Ref. [9:258].

Theorem IV.3 *Suppose that $X_1, X_2, \cdots, X_n$ are independent and each has a pdf Gamma (a_i, β). then the pdf of $X_1 + X_2 + \cdots + X_n$ is $Gamma(a_1 + a_2 + \cdots + a_n, \beta)$.*

Proof

Applying Theorem (IV.2) we have

$$\begin{aligned} M_{x_1+x_2+\cdots+x_n}(t) &= M_{x_1}(t)M_{x_2}(t)\cdots M_{x_n}(t) \\ &= (1-\beta t)^{-a1}(1-\beta t)^{-a2}\cdots(1-\beta t)^{-an} \\ &= (1-\beta t)^{-(a1+a2+\cdots+an)} \end{aligned}$$

i.e., $X_1 + X_2 + \cdots + X_n \sim Gamma(a_1 + a_2 + \cdots + a_n, \beta)$ --

Theorem IV.4 *If a random variable $X \sim Gamma(\alpha, \beta)$, then $cX \sim Gamma(\alpha, c\beta)$.*

Proof

Since $X \sim Gamma(\alpha, \beta)$, then its m.g.f. is

$$M_x(t) = (1 - \beta t)^{-\alpha}$$

Applying lemma (IV.2) to $M_{cx}(t)$ yields $M_x(ct)$ whose value is $(1 - \beta ct)^{-\alpha}$ which is the m.g.f. of Gamma $(\alpha, c\beta)$, i.e.,

$$cX \sim Gamma(\alpha, c\beta)$$

Corollary IV.1 *If X is a one-parameter exponential distribution, then*

$$cX \sim Gamma(1, c\sigma)$$

4.5 Order Statistics

Suppose that the random variable X has a continuous pdf $f(x)$ and that $\{X_1, X_2, \cdots X_n\}$ is a random sample from this distribution. The $X's$, rearranged in order of magnitude, and denoted

$$X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$$

are called the order statistics of the sample. A few results about order statistics will be given here.

Theorem IV.5 *Let $X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$ be the order statistics from a population with a pdf $EXP(\theta,\sigma)$ defined in Eq (4.24), then the pdf of X_1 is $EXP(\theta,\sigma/n)$*

Proof

$$\begin{aligned} P(X_{(1)} \leq x) &= 1-(1-F_x(x))^n & &= 1-(1-P(X_{(1)} \leq x))^n \\ &= 1-\left[1-1+e^{\frac{-(x-\theta)}{\sigma}}\right]^n & &= 1-e^{\frac{-n(x-\theta)}{\sigma}} \end{aligned}$$

The derivative of the above cdf of $X_{(1)}$ is

$$\frac{n}{\sigma}e^{[\frac{-n(x-\theta)}{\sigma}]} = \frac{1}{\sigma/n}e^{[\frac{-(x-\theta)}{\sigma/n}]}$$

Which means that $X_{(1)}$ has a two-parameter exponential distribution with parameters $\theta, \sigma/n$.

Theorem IV.6 *Let $X_{(1)} \leq X_{(2)} \leq \cdots \leq X_{(n)}$ be the order statistics from a population with a pdf $EXP(\sigma)$ defined in Eq (4.22) then the pdf of $X_{(1)}$ is $EXP(\sigma/n)$*

Proof

$$\begin{aligned} P(X_{(1)} \leq x) &= 1-(1-F_x(x))^n & &= 1-(1-1+e^{(-x/\sigma)})^n \\ &= 1-e^{\frac{-nx}{\sigma}} & &= 1-e^{\frac{-x}{\sigma/n}} \end{aligned}$$

Once again, the derivative of the cdf of $X_{(1)}$ is

$$\frac{1}{\sigma/n} e^{\frac{-x}{\sigma/n}}$$

which means that $X_{(1)}$ has a one-parameter exponential distribution with parameter σ/n.

Theorem IV.7 *let $X_1, X_2, \cdots, X_n$ be independent and identically distributed random variables (iid), with pdf $EXP(\sigma)$, then*

$$\sum_{i=1}^{n} X_i \sim Gamma(n, \sigma)$$

Proof

$X_i \sim EXP(\sigma)$ implies $X_i \sim Gamma(1, \sigma)$ by the definition of the exponential distribution, then by using theorem (IV.3) we get

$$\sum_{i=1}^{n} X_i \sim Gamma(\sum_{i=1}^{n} 1, \sigma)$$

i.e.,

$$\sum_{i=1}^{n} X_i \sim Gamma(n, \sigma) \tag{4.39}$$

Theorem IV.8 *if $X \sim Gamma(a, b)$ then $\frac{2X}{b} \sim \chi^2(2a)$*

Proof

$X \sim Gamma(a, b)$ implies $\frac{2X}{b} \sim Gamma(a, \frac{2b}{b}) \equiv Gamma(a, 2)$ by theory (IV.4) and this is equivalent to $\chi^2(2a)$ from the definition of the Chi-Squared distribution, i.e., $\frac{2X}{b} \sim \chi^2(2a)$.

Theorem IV.9 *If $X_1, X_2, \cdots, X_n$ are independent and identically distributed random variables with pdf $EXP(\sigma)$, then*

$$\frac{2T}{\sigma} \sim \chi^2(2n) \quad \text{where} \quad T = \sum_{i=1}^{n} X_i$$

Proof

Because $X_1, X_2, \cdots, X_n$ are independent and identically distributed random variables with pdf $EXP(\sigma)$, then $T \sim Gamma(n, \sigma)$ as given before, and by the last theorem, $\frac{2T}{\sigma}$ will be a Chi-Squared distribution with $2n$ degrees of freedom, that is

$$\frac{2T}{\sigma} \sim \chi^2(2n) \tag{4.40}$$

4.6 Type II Censoring

A type *II* censored sample is the one for which only the r smallest observations in a random sample of n items are observed $(1 \leq r \leq n)$. Experiments involving type *II* censoring are often used in many situations, such as software failures, where the test is terminated at the time of the $r^{\underline{th}}$ occurrence of failure. Such tests save time and money, since it could take a very long time for all failures to occur.

It should be clear that with type *II* censoring, the data consist of the r smallest times $X_{(1)} \leq X_{(2)} \leq ... \leq X_{(r)}$ out of a random sample of n times $X_1, X_2, ..., X_n$ from the distribution in question. If $X_1, X_2, ..., X_n$ are independent and identically distributed random variables, and have a continuous distribution with pdf $f(x)$, it follows from the general results of order statistics [47:518] that the joint pdf of $X_{(1)} \leq X_{(2)} \leq ... \leq X_{(r)}$ is

$$f(x_{(1)}, x_{(2)}, \ldots, x_{(r)}) = \frac{n!}{(n-r)!} \left(\prod_{i=1}^{r} f(x_{(i)}) \right) \left[1 - F(x_{(r)}) \right]^{n-r} \tag{4.41}$$

As an example, If $\{X_1, X_2, ..., X_n\}$ is a random sample from a one-parameter exponential distribution whose pdf is

$$f(x) = \frac{1}{\sigma}e^{-x/\sigma} \qquad x \geq 0, \sigma > 0$$

and knowing that $F(x) = 1 - e^{x/\sigma}$ i.e.

$$f(x_{(i)}) = \frac{1}{\sigma}e^{-x_{(i)}/\sigma}, \quad F(x_{(r)}) = 1 - e^{x_{(r)}/\sigma}$$

then the joint pdf of $X_{(1)} \leq X_{(2)} \leq ... \leq X_{(r)}$ is

$$\begin{aligned}
f(x_{(1)}, x_{(2)}, ..., x_{(r)}) &= \frac{n!}{(n-r)!}\left(\prod_{i=1}^{r}\frac{1}{\sigma}e^{-x_{(i)}/\sigma}\right)\left[1 - (1 - e^{-x_{(r)}/\sigma})\right]^{n-r} \\
&= \frac{n!}{(n-r)!}\frac{1}{\sigma^r}e^{-\frac{1}{\sigma}\sum_{i=1}^{r}x_{(i)}} \cdot e^{-(n-r)x_{(r)}/\sigma} \\
&= \frac{n!}{(n-r)!}\frac{1}{\sigma^r}e^{-\frac{1}{\sigma}\left[\sum_{i=1}^{r}x_{(i)} + (n-r)x_{(r)}\right]} \\
&= \frac{n!}{(n-r)!}\frac{1}{\sigma^r}e^{-T/\sigma}
\end{aligned}$$

where $T = \left[\sum_{i=1}^{r}x_{(i)} + (n-r)x_{(r)}\right]$ is called total time in the test whose pdf is $Gamma(r, \sigma)$ as can be proved by the following theorem.

Theorem IV.10 *The total time in the test T has a distribution which can be defined as*

$$\begin{aligned}
& T && \sim Gamma(r, \sigma) \\
or \quad & 2T && \sim Gamma(r, 2\sigma) \\
or \quad & 2T/\sigma && \sim Gamma(r, 2) \sim \chi^2_{(2r)}
\end{aligned}$$

Proof

Consider the following transformation

$$
\begin{aligned}
W_1 &= n x_{(1)} \\
W_2 &= (n-1)(x_{(2)} - x_{(1)}) \\
&\vdots \\
W_{r-1} &= (n-r+2)(x_{(r-1)} - x_{(r-2)}) \\
W_r &= (n-r+1)(x_{(r)} - x_{(r-1)})
\end{aligned}
$$

Adding the $W_i's$, gives

$$W_1 + W_2 + \cdots + W_r = \sum_{i=1}^{r} x_{(i)} + (n-r)x_{(r)}$$

i.e.,

$$\sum_{i=1}^{r} W_i = T$$

The joint pdf of $W_i's$ is given by [47:102]

$$f(w_1, w_2, ..., w_r) = f(x_{(1)}, x_{(2)}, ..., x_{(r)}) \left| \frac{\partial(x_{(1)}, x_{(2)}, ..., x_{(r)})}{\partial(W_1, W_2, ..., W_r)} \right|$$

where

$$\left| \frac{\partial(x_{(1)}, x_{(2)}, ..., x_{(r)})}{\partial(W_1, W_2, ..., W_r)} \right| = \frac{1}{\left| \frac{\partial(W_1, W_2, ..., W_r)}{\partial(x_{(1)}, x_{(2)}, ..., x_{(r)})} \right|}$$

Now

$$\left| \frac{\partial(W_1, W_2, ..., W_r)}{\partial(x_{(1)}, x_{(2)}, ..., x_{(r)})} \right| = \begin{vmatrix} \frac{\partial W_1}{\partial x_{(1)}} & \frac{\partial W_2}{\partial x_{(1)}} & \cdots & \frac{\partial W_r}{\partial x_{(1)}} \\ \frac{\partial W_1}{\partial x_{(2)}} & \frac{\partial W_2}{\partial x_{(2)}} & \cdots & \frac{\partial W_r}{\partial x_{(2)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial W_1}{\partial x_{(r)}} & \frac{\partial W_2}{\partial x_{(r)}} & \cdots & \frac{\partial W_r}{\partial x_{(r)}} \end{vmatrix}$$

Substituting, we get

$$\left|\frac{\partial(W_1, W_2, ..., W_r)}{\partial(x_{(1)}, x_{(2)}, ..., x_{(r)})}\right| = \begin{vmatrix} n & -(n-1) & \cdots & 0 \\ 0 & (n-1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (n-r+1) \end{vmatrix}$$

$$= \frac{n!}{(n-r)!}$$

Thus, $f(w_1, w_2, ..., w_r)$ can be expressed as

$$\begin{aligned} f(w_1, w_2, ..., w_r) &= \frac{n!}{(n-r)!}\frac{1}{\sigma^r}e^{-\frac{T}{\sigma}}\frac{1}{n!/(n-r)!} \\ &= \frac{1}{\sigma^r}e^{-\frac{T}{\sigma}} \\ &= \frac{1}{\sigma^r}e^{-\frac{\sum_{i=1}^{r} w_i}{\sigma}} \end{aligned}$$

which means that $W_1, W_2, ..., W_r$ are independent, each with a pdf

$$f(w_i) = \frac{1}{\sigma}e^{-\frac{w_i}{\sigma}}$$

Therefore, $T = \sum_{i=1}^{r} W_i$ can be defined by

$$\begin{aligned} & T && \sim Gamma(r, \sigma) \\ or\quad & 2T && \sim Gamma(r, 2\sigma) \\ or\quad & 2T/\sigma && \sim Gamma(r, 2) \quad \sim \chi^2_{(2r)} \end{aligned}$$

If the above discussion is to be applied for the two-parameter exponential distribution, then let $X_1, X_2, ..., X_n$ be a random sample from a two-parameter exponential distribution with a pdf

$$f(x;\theta,\sigma) = \begin{cases} \frac{1}{\sigma} e^{-\frac{x-\theta}{\sigma}} & x \geq \theta > 0, \sigma > 0 \\ 0 & \text{otherwise} \end{cases}$$

If $X_{(1)} \leq X_{(2)} \leq ... \leq X_{(r)}$ represents an order statistic from the above distribution, then the joint pdf of $X_{(1)}, X_{(2)}, ..., X_{(r)}$ is

$$\begin{aligned} f(x_{(1)}, x_{(2)}, ..., x_{(r)}) &= \tfrac{n!}{(n-r)!} \left(\prod_{i=1}^{r} \frac{1}{\sigma} e^{-(x_{(i)}-\theta)/\sigma} \right) \left[1 - (1 - e^{-(x_{(r)}-\theta)/\sigma})\right]^{n-r} \\ &= \tfrac{n!}{(n-r)!} \tfrac{1}{\sigma^r} e^{-(\frac{1}{\sigma})\left[\sum_{i=1}^{r}(x_{(i)}-\theta) + (n-r)(x_{(r)}-\theta)\right]} \end{aligned}$$

Theorem IV.11 *Let $X_{(1)} \leq X_{(2)} \leq ... \leq X_{(r)}$ be the order statistic from a two-parameter exponential distribution, then*

(a) $n(x_{(1)} - \theta)$ has a distribution given by $\frac{2n(x_{(1)}-\theta)}{\sigma} \sim \chi^2_{(2)}$

(b) Y_r has a distribution given by $\frac{2Y_r}{\sigma} \sim \chi^2_{2(r-1)}$ where

$$Y_r = \sum_{i=1}^{r} x_{(i)} + (n-r)x_{(r)} - nx_{(1)}$$

Proof

(a) It is already known that $X_{(1)}$ has a two-parameter exponential distribution $EXP(\theta, \sigma/n)$ and therefore $n(X_{(1)} - \theta)$ will have a one-parameter pdf $EXP(\sigma)$. This can also be written as

$$\frac{n(x_{(1)} - \theta)}{\sigma} \sim Gamma(1,1) \quad \text{or} \quad \frac{2n(x_{(1)} - \theta)}{\sigma} \sim Gamma(1,2) \quad \text{or} \quad \sim \chi^2_{(2)}$$

(b) since $(x_{(1)}-\theta) \le (x_{(2)}-\theta) \le \cdots \le (x_{(r)}-\theta)$ for all values of r, then the above statistics can be considered as the order statistic from a one-parameter exponential pdf $EXP(\sigma)$. Consider the following transformation

$$
\begin{aligned}
W_1 &= n(x_{(1)}-\theta) & & \\
W_2 &= (n-1)\left[(x_{(2)}-\theta)-(x_{(1)}-\theta)\right] &&= (n-1)(x_{(2)}-x_{(1)}) \\
&\vdots && \vdots \\
W_{r-1} &= (n-r+2)\left[(x_{(r-1)}-\theta)-(x_{(r-2)}-\theta)\right] &&= (n-r+2)(x_{(r-1)}-x_{(r-2)}) \\
W_r &= (n-r+1)\left[(x_{(r)}-\theta)-(x_{(r-1)}-\theta)\right] &&= (n-r+1)(x_{(r)}-x_{(r-1)})
\end{aligned}
$$

knowing that $W_1, W_2, ..., W_r$ are independent and each has a one-parameter exponential distribution, and that

$$
\begin{aligned}
Y_r &= \sum_{i=1}^{r} x_{(i)} + (n-r)x_{(r)} - nx_{(1)} \\
&= \sum_{i=1}^{r}(x_{(i)}-\theta) + (n-r)(x_{(r)}-\theta) - n(x_{(1)}-\theta) \\
&= \sum_{i=1}^{r} W_{(i)} - W_{(1)} = \sum_{i=2}^{r} W_{(i)}
\end{aligned}
$$

Since each $W_i \sim EXP(\sigma)$, then $\sum_{i=2}^{r} W_{(i)} \sim Gamma(r-1,\sigma)$

i.e., $Y_r \sim Gamma(r-1,\sigma)$ *or* $2Y/\sigma \sim Gamma(r-1,2) \sim \chi^2_{2(r-1)}$

4.7 Maximum Likelihood Estimates of θ and σ for the Two-Parameter Exponential Distribution

Let $X_{(1)} \le X_{(2)} \le ... \le X_{(r)}$ be the order statistic from a two-parameter exponential distribution, The likelihood function in a random sample of size n is then

$$
L(x_{(1)}, x_{(2)}, ..., x_{(r)}, \sigma, \theta) = \frac{n!}{(n-r)!}\frac{1}{\sigma^r} e^{-\frac{1}{\sigma}\left[\sum_{i=1}^{r}(x_{(i)}-\theta) + (n-r)(x_{(r)}-\theta)\right]} \tag{4.42}
$$

where $\theta \le x_{(1)} \le \cdots \le x_{(r)}$

To find the maximum likelihood estimator of σ and θ, one must first fix σ and maximize Eq (4.43) w.r.t. θ where θ varies over the range $\theta \leq x_{(1)}$. This means, θ must be selected so as to minimize

$$\frac{1}{\sigma}\left[\sum_{i=1}^{r}(x_{(i)}-\theta)+(n-r)(x_{(r)}-\theta)\right]=\frac{1}{\sigma}\left[\sum_{i=1}^{r}x_{(i)}+(n-r)x_{(r)}-n\theta\right]$$

It is clear that the minimum is attained at $\widehat{\theta}=x_{(1)}$.

For $\widehat{\sigma}$, let us consider

$$L(x_{(1)},x_{(2)},\ldots,x_{(r)},\sigma,\theta)=\left(\frac{n!}{(n-r)!}\right)\frac{1}{\sigma^{r}}e^{-\frac{1}{\sigma}Y_r}$$

where $\quad Y_r=\left(\sum_{i=1}^{r}x_{(i)}+(n-r)x_{(r)}-nx_{(1)}\right)$, $\quad$ then

$$\ell nL(x_{(1)},x_{(2)},\ldots,x_{(r)},\sigma,\theta)=\ell n\left(\frac{n!}{(n-r)!}\right)-r\ell n\sigma-\frac{1}{\sigma}Y_r$$

differentiating w.r.t. σ, we get

$$\frac{\partial\ell nL}{\partial\sigma}=0-\frac{r}{\sigma}+\frac{Y_r}{\sigma^2}$$

putting $\quad \frac{\partial\ell nL}{\partial\sigma}=0 \quad$ gives $\sigma=\frac{Y_r}{r}$, i.e. the M.L.E. of σ is $\widehat{\sigma}=\left(\frac{Y_r}{r}\right)$.

4.8 The Likelihood Ratio Test (LRT) Principle

Let $X_1, X_2, \ldots, X_n$ denote n independent random variables having respectively the probability density function $f_i(x_{i;\theta_1,\theta_2,\cdots,\theta_m})$, $i=1,2,\cdots,n$. The set that consists of all parameter points $(\theta_1,\theta_2,\cdots,\theta_m)$ is denoted by Ω, which is called the parameter space. Let ω be a subset of the parameter space Ω. If ω is the set of unknown parameter values admissible under H_o, then we wish to test the hypothesis:

$$H_o:\ (\theta_1,\theta_2,\cdots,\theta_m)\in\omega \qquad \text{against} \qquad H_a:\ (\theta_1,\theta_2,\cdots,\theta_m)\notin\omega$$

Define the likelihood functions

$$\begin{aligned} L(\omega) &= \prod_{i=1}^{n} f_i(x_{i;\theta_1,\theta_2,\cdots,\theta_m}), & \theta_1,\theta_2,\cdots,\theta_m \in \omega \\ \text{and} \quad L(\Omega) &= \prod_{i=1}^{n} f_i(x_{i;\theta_1,\theta_2,\cdots,\theta_m}), & \theta_1,\theta_2,\cdots,\theta_m \in \Omega \end{aligned}$$

Let $L(\hat{\omega})$ and $L(\hat{\Omega})$ be the maxima of these two likelihood functions, which are assumed to exist. The ratio $\frac{L(\hat{\omega})}{L(\hat{\Omega})}$ is called the *likelihood ratio* and is denoted by

$$\Lambda = \frac{L(\hat{\omega})}{L(\hat{\Omega})}$$

Let λ^* be a positive proper constant. The ***Likelihood ratio test principle*** states that the hypothesis $H_o : (\theta_1, \theta_2, \cdots, \theta_m) \in \omega$ is rejected if and only if

$$\Lambda \leq \lambda^*$$

λ^* is determined according to the value of the significance level of the test α according to the following equation.

$$P(\Lambda \leq \lambda^* \mid H_o \quad is \ true) = \alpha$$

As an example, if the level of significance is 0.05, then there is a 5% probability that the tested hypothesis will be rejected when it is actually true. In this research, a 5% chance of mistakenly saying that the MTBF 's of the samples are not the same.

Our objective in the next section is to obtain the cumulative distribution function of the likelihood ratio, in a computational form so that the value of λ^* can be found for any level of significance, any number of samples, and any number of failures.

4.9 *Derivation of the Likelihood Ratio Criterion*

Suppose that there are p populations with the pdf of the $i^{\underline{th}}$ population be $EXP(\theta_i, \sigma_i)$ and it is required to find the likelihood ratio criterion for testing

$$\begin{aligned} H_o \quad &: \quad \theta_1 = \theta_2 = \cdots = \theta_p = \theta \quad \text{(unknown)} \\ \& \quad & \quad \sigma_1 = \sigma_2 = \cdots = \sigma_p = \sigma \quad \text{(unknown)} \\ \text{against } H_a \quad &: \quad \text{otherwise.} \end{aligned}$$

$X_{1(1)}$ $X_{1(2)}$ $\vdots$ $X_{1(r_1)}$	$\cdots$	$X_{i(1)}$ $X_{i(2)}$ $\vdots$ $X_{i(r_i)}$	$\cdots$	$X_{p(1)}$ $X_{p(2)}$ $\vdots$ $X_{p(r_p)}$
Pop. 1 $EXP(\theta_1, \sigma_1)$		Pop. i $EXP(\theta_i, \sigma_i)$		Pop. p $EXP(\theta_p, \sigma_p)$

Let $\quad Z_i = min(X_{1(1)}, X_{2(1)}, \cdots, X_{i(1)})$

so $\quad Z_p = min(X_{1(1)}, X_{2(1)}, \cdots, X_{p(1)})$

Theorem IV.12

$$Let \quad Y_i = \sum_{j=1}^{r_i}(X_{i(j)} - X_{i(1)}) + (n_i - r_i)(X_{i(r_i)} - X_{i(1)}) \tag{4.43}$$

$$, \quad V = \sum_{i=1}^{p}\sum_{j=1}^{r_i}(X_{i(j)} - Z_p) + \sum_{i=1}^{p}(n_i - r_i)(X_{i(r_i)} - Z_p) \tag{4.44}$$

$$and \quad U_i = (\sum_{j=1}^{i-1} n_j)Z_{i-1} + n_i X_{i(1)} - (\sum_{j=1}^{i} n_j)Z_i \tag{4.45}$$

$$where \qquad i = 2, 3, \cdots$$

When H_o is true, we will have

(a). $\sum_{i=1}^{p} Y_i \sim Gamma(R - p, \sigma)$

(b). $V \sim Gamma(R - 1, \sigma)$

(c). $V = \sum_{i=1}^{p} Y_i + u$

where $$R = \sum_{i=1}^{p} r_i \quad and \quad u = \sum_{i=2}^{p} U_i$$

Proof

(a). It is already proved that $Y_i \sim Gamma(r_i - 1, \sigma_i)$, and therefore if H_o is true, then $Y_i \sim Gamma(r_i - 1, \sigma)$. Since the $Y_i's$ are independent, then $\sum_{i=1}^{p} Y_i \sim Gamma(\sum_{i=1}^{p}(r_i - 1), \sigma) \sim Gamma(R - p, \sigma)$.

(b). Since Z_p is the smallest among $\{X_{i(j)} \mid i = 1, \cdots, p; \quad j = 1, \cdots, r_i\}$ so, using the same result of (a), we conclude that $V \sim Gamma(\sum_{i=1}^{p}(r_i) - 1, \sigma)$ i.e., $V \sim Gamma(R - 1, \sigma)$.

(c). From the definition of U_i, one can write

$$\begin{aligned}
U_2 &= (\sum_{j=1}^{1} n_j) Z_1 + n_2 X_{2(1)} - (\sum_{j=1}^{2} n_j) Z_2 \\
&= n_1 Z_1 + n_2 X_{2(1)} - (n_1 + n_2) Z_2 \\
U_3 &= (n_1 + n_2) Z_2 + n_3 X_{3(1)} - (n_1 + n_2 + n_3) Z_3 \\
&\vdots \\
U_{p-1} &= (n_1 + n_2 + \cdots + n_{p-2}) Z_{p-2} + n_{p-1} X_{p-1(1)} - (n_1 + n_2 + \cdots + n_{p-1}) Z_{p-1} \\
U_p &= (n_1 + n_2 + \cdots + n_{p-1}) Z_{p-1} + n_p X_{p(1)} - (n_1 + n_2 + \cdots + n_p) Z_p
\end{aligned}$$

by summation, we get

$$\begin{aligned}
\sum_{i=2}^{p} U_i &= n_1 Z_1 + n_2 X_{2(1)} + \cdots + n_p X_{p(1)} - (n_1 + n_2 + \cdots + n_p) Z_p \\
&= n_1(Z_1 - Z_p) + n_2(X_{2(1)} - Z_p) + \cdots + n_p(X_{p(1)} - Z_p) \\
&= n_1(X_{1(1)} - Z_p) + \sum_{i=2}^{p} n_i(X_{i(1)} - Z_p) \quad = \quad \sum_{i=1}^{p} n_i(X_{i(1)} - Z_p) \qquad (4.46)
\end{aligned}$$

Eq (4.44) for V can be written as

$$\begin{aligned}
V &= \sum_{i=1}^{p}\sum_{j=1}^{r_i}(X_{i(j)} - Z_p) + \sum_{i=1}^{p}(n_i - r_i)(X_{i(r_i)} - Z_p) \\
&= \sum_{i=1}^{p}\sum_{j=1}^{r_i}(X_{i(j)} - X_{i(1)} + X_{i(1)} - Z_p) + \sum_{i=1}^{p}(n_i - r_i)(X_{i(r_i)} - X_{i(1)} + X_{i(1)} - Z_p) \\
&= \sum_{i=1}^{p}\sum_{j=1}^{r_i}(X_{i(j)} - X_{i(1)}) + \sum_{i=1}^{p}(n_i - r_i)(X_{i(r_i)} - X_{i(1)}) \\
&\qquad + \sum_{i=1}^{p}\sum_{j=1}^{r_i}(X_{i(1)} - Z_p) + \sum_{i=1}^{p}(n_i - r_i)(X_{i(1)} - Z_p) \\
&= \sum_{i=1}^{p} Y_i + u \qquad (4.47)
\end{aligned}$$

where Y_i is as given in (4.43), and

$$\begin{aligned}
u &= \sum_{i=1}^{p}\sum_{j=1}^{r_i}(X_{i(1)} - Z_p)\sum_{i=1}^{p}(n_i - r_i)(X_{i(1)} - Z_p) \\
&= \sum_{i=1}^{p} r_i(X_{i(1)} - Z_p) + \sum_{i=1}^{p} n_i(X_{i(1)} - Z_p) - \sum_{i=1}^{p} r_i(X_{i(1)} - Z_p) \\
&= \sum_{i=1}^{p} n_i(X_{i(1)} - Z_p)
\end{aligned}$$

which according to Eq (4.46) equals $\sum_{i=2}^{p} U_i$

$$\text{i.e.} \qquad V = \sum_{i=1}^{p} Y_i + u \quad \text{where} \qquad u = \sum_{i=2}^{p} U_i \qquad (4.48)$$

Theorem IV.13 *If H_o is true, then* $u \sim Gamma(p-1, \sigma)$.

Proof

Hogg and Tanis [35:439], proved that $U_i \sim Gamma(1, \sigma)$ and since they are independent, then $U_2 + U_3 + \cdots + U_p \sim Gamma(p-1, \sigma)$, and therefore $u \sim Gamma(p-1, \sigma)$.

The last two theorems will be used again in the next chapter for the derivation of the h^{th} moment of the likelihood ratio criterion Λ.

For the derivation of the likelihood ratio criterion, remember that during the discussion about the Maximum Likelihood Estimates, the M.L.E. of θ_i was $X_{i(1)}$ and that of σ_i was $(\frac{Y_i}{r_i})$ where

$$Y_i = \sum_{j=1}^{r_i}(X_{i(j)} - X_{i(1)}) + (n_i - r_i)(X_{i(r_i)} - X_{i(1)})$$

The likelihood function of $X_{1(1)}, \cdots, X_{p(r_p)}$ is

$$L(X_{1(1)}, \cdots, X_{p(r_p)}; \theta_i; \sigma_i) = k \cdot \prod_{i=1}^{p} \frac{1}{\sigma_i^{r_i}} \cdot e^{-\frac{1}{\sigma_i}\left[\sum_{j=1}^{r_i}(x_{i(j)} - \theta_i) + (n_i - r_i)(x_{i(r_i)} - \theta_i)\right]}$$

where $k = \prod_{i=1}^{p} \frac{n_i!}{(n_i - r_i)!}$

Under the null hypothesis H_o

$$\theta_1 = \theta_2 = \cdots = \theta_p = \theta, \quad \text{and} \quad \sigma_1 = \sigma_2 = \cdots = \sigma_p = \sigma$$

so, the likelihood function can be written as

$$\begin{aligned} L(X_{1(1)}, \cdots, X_{p(r_p)}; \theta_i; \sigma_i) &= k \cdot \prod_{i=1}^{p} \frac{1}{\sigma^{r_i}} \cdot e^{-\frac{1}{\sigma}\left[\sum_{j=1}^{r_i}(X_{i(j)} - \theta) + (n_i - r_i)(X_{i(r_i)} - \theta)\right]} \\ &= k\frac{1}{\sigma^R} \cdot e^{-\frac{1}{\sigma}\left[\sum_{i=1}^{p}\sum_{j=1}^{r_i}(X_{i(j)} - \theta) + \sum_{i=1}^{p}(n_i - r_i)(X_{i(r_i)} - \theta)\right]} \end{aligned} \tag{4.49}$$

The M.L.E. of θ is $Z_p = \min(\{x_{i(j)} \mid i = 1, 2, \cdots, p; \quad j = 1, 2, \cdots, r_i\})$.

Substituting in Eq (4.49) using (4.44), we get

$$L(X_{1(1)},\cdots,X_{p(r_p)};\theta_i;\sigma_i) = k\cdot\frac{1}{\sigma^R}\cdot e^{-(\frac{V}{\sigma})} \tag{4.50}$$

The M.L.E. of σ is then $\hat{\sigma} = \frac{V}{R}$, so

$$\begin{aligned}\max_{\theta_i,\sigma_i\in H_o} L(X_{1(1)},\cdots,X_{p(r_p)};\theta_i;\sigma_i) &= k\cdot\frac{1}{(V/R)^R}\cdot e^{-\frac{V}{(V/R)}}\\ &= k\cdot\frac{R^R}{V^R}\cdot e^{-R} \end{aligned}\tag{4.51}$$

when θ_i and σ_i are unrestricted, the M.L.E. of θ_i is $X_{i(1)}$ and the M.L.E. of σ_i is $(\frac{Y_i}{r_i})$ so

$$\max_{\theta_i,\sigma_i\ unrest.} L(X_{1(1)},\cdots,X_{p(r_p)};\theta_i;\sigma_i) = k\prod_{i=1}^{p}\frac{1}{(Y_i/r_i)^{r_i}}e^{-\frac{1}{(Y_i/r_i)}\left[\sum_{j=1}^{r_i}(X_{i(j)}-X_{i(1)})+(n_i-r_i)(X_{i(r_i)}-X_{i(1)})\right]}$$

$$\begin{aligned} &= k\prod_{i=1}^{p}\frac{(r_i)^{r_i}}{(Y_i)^{r_i}}\cdot e^{-\frac{r_i}{Y_i}\cdot Y_i} = k\prod_{i=1}^{p}\frac{(r_i)^{r_i}}{(Y_i)^{r_i}}\cdot e^{-r_i}\\ &= k\left(\prod_{i=1}^{p}\frac{(r_i)^{r_i}}{(Y_i)^{r_i}}\right)\cdot e^{-\sum_{i=1}^{p}r_i} = k\left(\prod_{i=1}^{p}\frac{(r_i)^{r_i}}{(Y_i)^{r_i}}\right)\cdot e^{-R} \end{aligned}\tag{4.52}$$

From (4.51) and (4.52), the likelihood ratio is given by

$$\begin{aligned}\Lambda &= \frac{\max_{\theta_i,\sigma_i\in H_o} L(X_{1(1)},\cdots,X_{p(r_p)};\theta_i;\sigma_i)}{\max_{\theta_i,\sigma_i unrestricted} L(X_{1(1)},\cdots,X_{p(r_p)};\theta_i;\sigma_i)}\\ &= \frac{k\cdot\frac{R^R}{V^R}\cdot e^{-R}}{k\cdot\left(\prod_{i=1}^{p}\frac{(r_i)^{r_i}}{(Y_i)^{r_i}}\right)\cdot e^{-R}} = \frac{(\frac{R}{V})^R}{\prod_{i=1}^{p}(\frac{r_i}{Y_i})^{r_i}}\\ &= \frac{R^R}{V^R}\cdot\frac{\prod_{i=1}^{p}(Y_i)^{r_i}}{\prod_{i=1}^{p}(r_i)^{r_i}} = \frac{R^R}{(u+\sum_{i=1}^{p}Y_i)^R}\cdot\frac{\prod_{i=1}^{p}(Y_i)^{r_i}}{\prod_{i=1}^{p}(r_i)^{r_i}} \end{aligned}\tag{4.53}$$

V. THE EXACT DISTRIBUTION OF THE TEST STATISTIC

For our approach of selecting the proper software reliability model, a test statistic is needed. This statistic will be used for testing, whether there are significant differences among various sets of software failure data. It has been taken into account that the underlying distribution is a two-parameter exponential with type *II* censoring, and the failure data sets are of unequal sizes. Nagarsenker and Nagarsenker [70] obtained the exact distribution of Likelihood Ratio Test (LRT) for testing the equality of two or more exponential distributions using, equal uncensored sets of failure data. For the equality of one-parameter exponential distributions see Cole and others [39].

In this chapter, the exact distribution of the Likelihood Ratio Test (LRT), based on unequal type *II* censored sets of failure data, is obtained for the first time, in a computational form.

5.1 Assumptions

1. p independent, type *II* censored sets of failure data are available, each has a two-parameter exponential distribution given by

$$f(x;\theta_i,\sigma_i) = \begin{cases} \frac{1}{\sigma_i} \cdot e^{-\frac{x-\theta_i}{\sigma_i}} & x \geq \theta_i, \sigma_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

where $i = 1, 2, \cdots, p$, is a subscript denoting data set i, p is the number of data sets. θ_i is the location parameter and σ_i is the scale parameter.

2. Each set has an unequal number of observations.

The LRT for testing the hypothesis

$$H_o \quad : \quad \theta_1 = \theta_2 = \cdots = \theta_p \quad \text{and} \quad \sigma_1 = \sigma_2 = \cdots = \sigma_p \tag{5.2}$$

against the general alternatives based on the ordered sets of data where $r_i \leq n_i$ was given in the

previous chapter by

$$\Lambda = \frac{R^R}{V^R} \cdot \frac{\prod_{i=1}^{p}(Y_i)^{r_i}}{\prod_{i=1}^{p}(r_i)^{r_i}} = \frac{R^R}{(u+\sum_{i=1}^{p}Y_i)^R} \cdot \frac{\prod_{i=1}^{p}(Y_i)^{r_i}}{\prod_{i=1}^{p}(r_i)^{r_i}} \tag{5.3}$$

where

H_o	=	Null Hypothesis
Λ	=	likelihood ratio
r_i	=	size of censored sample i; $1 \leq r_i \leq n_i$
R	=	$\sum_{i=1}^{p} r_i$, total number of failures
Y_i	=	$\sum_{j=1}^{r_i}(x_{i(j)} - x_{i(1)}) + (n_i - r_i)(x_{i(r_i)} - x_{i(1)})$
n_i	=	sample size
V	=	$\sum_{i=1}^{p}\sum_{j=1}^{r_i}(x_{i(j)} - Z_p) + \sum_{i=1}^{p}(n_i - r_i)(x_{i(r_i)} - Z_p)$
Z_p	=	$min(X_{1(1)}, X_{2(1)}, \cdots, X_{p(1)})$

5.2 Derivation of the $h^{\underline{th}}$ Moment of Λ

To get the $h^{\underline{th}}$ moment of Λ, we use the method described by Wilks [90:391], and the fact that under H_o :

$$\theta_1 = \theta_2 = \cdots = \theta_p = \theta \tag{5.4}$$

$$\& \quad \sigma_1 = \sigma_2 = \cdots = \sigma_p = \sigma \tag{5.5}$$

and that $Y_i \sim Gamma(r_i - 1, \sigma)$, and $u \sim Gamma(p-1, \sigma)$ as given in the previous chapter.

Note: From Hogg and Tanis [35:439], U_i and Y_i are independent for each i, and so $\sum_{i=1}^{p} Y_i$ and $u = \sum_{i=2}^{p} U_i$ are also independent.

Theorem V.1 *Let $Y_i \sim Gamma(\alpha_i, \beta)$, $i = 1, 2, \cdots, p$, and $W \sim Gamma(k, \beta)$. Further, assume that $\sum_{i=1}^{p} Y_i$ and W are independent, then the $h^{\underline{th}}$ moment of*

$$Y = \frac{\prod_{i=1}^{p}(Y_i)^{r_i}}{(\sum_{i=1}^{p} Y_i + W)^R} \tag{5.6}$$

where $r_i > 0$ are constants and $R = r_1 + r_2 + \cdots + r_p$ is given by

$$E(Y^h) = \frac{\Gamma(\sum_{i=1}^{p} \alpha_i + k)}{\Gamma(\sum_{i=1}^{p} \alpha_i + k + Rh)} \prod_{i=1}^{p} \frac{\Gamma(r_i h + \alpha_i)}{\Gamma(\alpha_i)} \tag{5.7}$$

Proof

Consider the function $f(\theta)$ where

$$f(\theta) = E\left[e^{\theta(w + \sum_{i=1}^{p} y_i)} \prod_{i=1}^{p} (y_i)^{r_i h} \right] \tag{5.8}$$

Following the method of Wilks [90:391], the $h^{\underline{th}}$moment of Y is

$$E(Y^h) = \left. \frac{d^s f(\theta)}{d\theta^s} \right|_{\substack{\theta=0 \\ s=-Rh}} \tag{5.9}$$

Now

$$f(\theta) \quad = \quad E\left[e^{\theta(w + \sum_{i=1}^{p} y_i)} \prod_{i=1}^{p} (y_i)^{r_i h} \right]$$

which can be expressed as

$$f(\theta) = \int_{w>0}^{\infty}\int_{y_1>0}^{\infty}\cdots\int_{y_p>0}^{\infty}\prod_{i=1}^{p}(y_i)^{r_i h} e^{\theta(w+\sum_{i=1}^{p} y_i)} f(y_1, y_2, \cdots, y_p, w)dy_1 dy_2\cdots dy_p dw \quad (5.10)$$

where $f(y_1, y_2, \cdots, y_p, w)$ is the joint pdf of $Y_1, Y_2, \cdots, Y_p$ and W.

Since $Y_i's$ and W are independent, then we have

$$f(y_1, y_2, \cdots, y_p, w) = f(y_1)\cdot f(y_2)\cdots f(y_p)\cdot f(w) \quad (5.11)$$

further

$$f(y_i) = Gamma(\alpha_i, \beta) = \frac{1}{\beta^{\alpha_i}\Gamma(\alpha_i)} y_i^{\alpha_i - 1} e^{-y_i/\beta} \quad (5.12)$$

and similarly

$$f(w) = \frac{1}{\beta^k\Gamma(k)} w^{k-1} e^{-w/\beta} \quad (5.13)$$

The joint pdf $f(y_1, y_2, \cdots, y_p, w)$ can be expressed as

$$f(y_1, y_2, \cdots, y_p, w) = \frac{1}{\beta^k\Gamma(k)} w^{k-1} e^{-w/\beta} \prod_{i=1}^{p} \frac{1}{\beta^{\alpha_i}\Gamma(\alpha_i)} y_i^{\alpha_i - 1} e^{-y_i/\beta} \quad (5.14)$$

substituting for $f(\theta)$, we get

$$\begin{aligned} f(\theta) &= \int_{w>0}^{\infty}\int_{y_1>0}^{\infty}\cdots\int_{y_p>0}^{\infty}\left[\prod_{i=1}^{p}(y_i)^{r_i h}\right] e^{\theta(w+\sum_{i=1}^{p} y_i)} \\ &\quad \cdot\left\{\prod_{i=1}^{p}\left[\frac{1}{\beta^{\alpha_i}\Gamma(\alpha_i)} y_i^{\alpha_i - 1} e^{-y_i/\beta}\right]\right\} \frac{1}{\beta^k\Gamma(k)} w^{k-1} e^{-w/\beta} dy_1 dy_2\cdots dy_p dw \\ &= \frac{1}{\beta^k\Gamma(k)(\prod_{i=1}^{p}\beta^{\alpha_i})\prod_{i=1}^{p}\Gamma(\alpha_i)} \int_{w>0}^{\infty} w^{k-1} e^{-w(\frac{1}{\beta}-\theta)} dw \\ &\quad \prod_{i=1}^{p}\int_{y_i>0}^{\infty}\left[y_i^{\alpha_i + r_i h - 1} e^{-y_i(\frac{1}{\beta}-\theta)}\right] dy_i \end{aligned} \quad (5.15)$$

Taking into account that $\int_0^\infty x^p e^{\alpha x} dx = \frac{\Gamma(p+1)}{\alpha^{p+1}}$, the last equation can be written as

$$
\begin{aligned}
f(\theta) &= \frac{1}{\beta^k \Gamma(k) \cdot \beta^{\sum_{i=1}^{p} \alpha_i} \cdot \prod_{i=1}^{p} \Gamma(\alpha_i)} \cdot \frac{\Gamma(k)}{(\frac{1}{\beta} - \theta)^k} \prod_{i=1}^{p} \frac{\Gamma(\alpha_i + r_i h)}{(\frac{1}{\beta} - \theta)^{\alpha_i + r_i h}} \\
&= \frac{1}{\beta^k \Gamma(k) \beta^{\sum_{i=1}^{p} \alpha_i} \prod_{i=1}^{p} \Gamma(\alpha_i)} \cdot \frac{\beta^k \Gamma(k)}{(1-\beta\theta)^k} \frac{\prod_{i=1}^{p} \Gamma(\alpha_i + r_i h)}{(1-\beta\theta)^{\sum_{i=1}^{p} \alpha_i + Rh}} \beta^{\sum_{i=1}^{p} \alpha_i + Rh} \\
&= \frac{\beta^{Rh}}{(1-\beta\theta)^{\sum_{i=1}^{p} \alpha_i + Rh + k}} \prod_{i=1}^{p} \frac{\Gamma(\alpha_i + r_i h)}{\Gamma(\alpha_i)} \\
&= \beta^{Rh} \prod_{i=1}^{p} \frac{\Gamma(\alpha_i + r_i h)}{\Gamma(\alpha_i)} (1-\beta\theta)^{-(\sum_{i=1}^{p} \alpha_i + Rh + k)} \\
&= K(1-\beta\theta)^{-\alpha} \qquad (5.16)
\end{aligned}
$$

where

$$
K = \beta^{Rh} \prod_{i=1}^{p} \frac{\Gamma(\alpha_i + r_i h)}{\Gamma(\alpha_i)} \quad and \quad \alpha = (\sum_{i=1}^{p} \alpha_i + Rh + k) \qquad (5.17)
$$

differentiating,we get

$$
\begin{aligned}
\frac{d^s}{d\theta^s} f(\theta) &= \frac{d^s}{d\theta^s} K(1-\beta\theta)^{-\alpha} \\
&= K\alpha(\alpha+1)(\alpha+2)\cdots(\alpha+s-1)\beta^s(1-\beta\theta)^{-(\alpha-s)} \\
&= K\frac{\Gamma(\alpha+s)}{\Gamma(\alpha)}\beta^s(1-\beta\theta)^{-(\alpha-s)} \qquad (5.18)
\end{aligned}
$$

At $\theta = 0$, $\left.\frac{d^s f(\theta)}{d\theta^s}\right|_{\theta=0} = K\frac{\Gamma(\alpha+s)}{\Gamma(\alpha)}\beta^s$ i.e.

$$\left.\frac{d^s f(\theta)}{d\theta^s}\right|_{\theta=0} = \beta^{Rh}\left[\prod_{i=1}^{p}\frac{\Gamma(\alpha_i + r_i h)}{\Gamma(\alpha_i)}\right]\frac{\Gamma(\sum_{i=1}^{p}\alpha_i + Rh + k + s)}{\Gamma(\sum_{i=1}^{p}\alpha_i + Rh + k)}\beta^s \tag{5.19}$$

Putting $s = -Rh$, we have

$$\left.\frac{d^s f(\theta)}{d\theta^s}\right|_{\substack{\theta=0\\ s=-Rh}} = \beta^{Rh}\left[\prod_{i=1}^{p}\frac{\Gamma(\alpha_i + r_i h)}{\Gamma(\alpha_i)}\right]\frac{\Gamma(\sum_{i=1}^{p}\alpha_i + Rh + k - Rh)}{\Gamma(\sum_{i=1}^{p}\alpha_i + Rh + k)}\beta^{-Rh}$$

$$i.e.\ \ E(Y^h) = \frac{\Gamma(\sum_{i=1}^{p}\alpha_i + k)}{\Gamma(\sum_{i=1}^{p}\alpha_i + Rh + k)}\prod_{i=1}^{p}\frac{\Gamma(\alpha_i + r_i h)}{\Gamma(\alpha_i)} \tag{5.20}$$

Theorem V.2 *Under the null hypothesis H_o, the $h^{\underline{th}}$ moment of Λ defined in (5.3) is given by*

$$E(\Lambda^h) = \frac{R^{Rh}\Gamma(R-1)}{\Gamma(R(1+h)-1)}\prod_{i=1}^{p} r_i^{-r_i h}\frac{\Gamma(r_i(1+h)-1)}{\Gamma(r_i - 1)}, \tag{5.21}$$

Proof

In the previous chapter, we have seen that Λ can be written as

$$\Lambda = \frac{R^R}{\prod_{i=1}^{p}(r_i)^{r_i}}\frac{\prod_{i=1}^{p}(Y_i)^{r_i}}{(u + \sum_{i=1}^{p} Y_i)^R} \tag{5.22}$$

and taking into consideration that $Y_i \sim Gamma(r_i - 1, \sigma)$, and $u \sim Gamma(p-1, \sigma)$ and applying Theorem (V.1), we have

$$\Lambda^h = \frac{R^{Rh}}{\left(\prod_{i=1}^{p}(r_i)^{r_i}\right)^h}\left(\frac{\prod_{i=1}^{p}(Y_i)^{r_i}}{(u + \sum_{i=1}^{p} Y_i)^R}\right)^h \tag{5.23}$$

Therefore

$$
\begin{aligned}
E(\Lambda^h) &= \frac{R^{Rh}}{\left(\prod_{i=1}^{p}(r_i)^{r_i}\right)^h} \cdot E\left(\frac{\prod_{i=1}^{p}(Y_i)^{r_i}}{(u+\sum_{i=1}^{p}Y_i)^R}\right)^h \\
&= \frac{R^{Rh}}{\left(\prod_{i=1}^{p}(r_i)^{r_i}\right)^h} \frac{\Gamma(\sum_{i=1}^{p}(r_i-1)+p-1)}{\Gamma(\sum_{i=1}^{p}(r_i-1)+p-1+Rh)} \prod_{i=1}^{p} \frac{\Gamma(r_i h + r_i - 1)}{\Gamma(r_i - 1)} \\
&= \frac{R^{Rh}\cdot\Gamma(R-p+p-1)}{\Gamma(R-p+p-1+Rh)} \prod_{i=1}^{p} \frac{\Gamma(r_i(h+1)-1)}{\Gamma(r_i-1)}(r_i)^{-r_i h} \\
&= \frac{R^{Rh}\cdot\Gamma(R-1)}{\Gamma(R(1+h)-1)} \prod_{i=1}^{p} r_i^{-r_i h} \frac{\Gamma(r_i(1+h)-1)}{\Gamma(r_i-1)} \qquad (5.24)
\end{aligned}
$$

Having defined the test statistic and its $h^{\underline{th}}$ moment, it is time now to know the probability distribution for its values, and have the cumulative distribution function from which the expected value of the statistic for a given probability can be calculated. In order to do that, the Mellin Inverse Transform will be used to obtain the probability distribution from the moment generating function which already has been derived. In order to obtain a computable form of the cumulative distribution function of the statistic, the asymptotic expansion of the Gamma function will be used. Before proceeding, the following results will be needed: *Gamma Expansion:*

The following expansion for the natural logarithm of the Gamma function holds [5:312]

$$\ell n\Gamma(x+h) = \ell n(2\pi)^{\frac{1}{2}} + (x+h-\frac{1}{2})\ell n x - x - \sum_{r=1}^{m}\frac{(-1)^r B_{r+1}(h)}{r(r+1)x^r} + R_m(x) \qquad (5.25)$$

where $R_m(x)$ is the remainder, such that $|R_m(x)| \leq \frac{C}{|x^m|}$ for some constant C independent of x. and $B_r(h)$ is the Bernoulli Polynomial of degree r and order unity given by the following relation

$$\frac{te^{ht}}{e^t-1} = \sum_{r=0}^{\infty}(\frac{t^r}{r!})B_r(h) \qquad (5.26)$$

The first three Polynomials are [5:312]

$$B_1(h) = h - \frac{1}{2} \tag{5.27}$$

$$B_2(h) = h^2 - h + \frac{1}{6} \tag{5.28}$$

$$B_3(h) = h^3 - (\frac{3}{2})h^2 + (\frac{1}{2})h \tag{5.29}$$

Inverse Mellin Transform:

If $\phi(h) = E(x^h)$, then the pdf of $x > 0$ is the inverse Mellin Transform given by [88]

$$f(x) = \frac{1}{2\pi i}\int_{-i\infty}^{i\infty} x^{-h-1}\phi(h)dh \tag{5.30}$$

$Q(t)$:

$f(t)$ is $Q(t)$ if the function $f(t)$ is bounded by some constant multiple of t for large t. *Nair & Norlunds Result* [72],[74]:

If $\phi(h) = Q(h^{-k})_{h\to\infty}$, i.e. $\mid \frac{\phi(h)}{h^k} \mid$ is bounded; then $\phi(h)$ can be expanded as a factorial series in the form

$$\phi(h) = \sum_{i=0}^{\infty} R_i \frac{\Gamma(h+a)}{\Gamma(h+k+i+a)} \tag{5.31}$$

where $a \geq 0$ is an arbitrary constant chosen such that $R_1 = 0$ and the coefficients R_i are obtained using the following recurrence relations from [71:363]

$$\sum_{j=0}^{i} R_{i-j}d_{i-j,j} = q_i, \quad i = 1, 2, 3, \cdots \tag{5.32}$$

$$d_{i,r} = \sum_{k=1}^{r} kc_{i,k}d_{i,r-k}/r, \qquad d_{i,0=1} \tag{5.33}$$

$$c_{i,r} = \frac{(-1)^{r-1}}{r(r+1)}[B_{r+1}(a) - B_{r+1}(a+v+i)] \tag{5.34}$$

Beta Function:

The Beta function of w and z is defined as

$$B(w,z) \quad = \quad B(z,w) = \int_0^1 t^{z-1}(1-z)^{w-1}dt \tag{5.35}$$

which is related to the Gamma function by

$$B(z,w) \quad = \quad \frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)} \tag{5.36}$$

Incomplete Beta Function:

The Incomplete Beta Function $I_x(a,b)$ is defined by

$$I_x(a,b) = \frac{1}{B(a,b)} \int_0^x t^{a-1}(1-t)^{b-1}dt \qquad (a,b>0) \tag{5.37}$$

5.3 The Null Distribution of Λ

Let $L = \Lambda^{p/R}$, then from (5.24) we have

$$E(L^h) = K \cdot \frac{R^{ph}}{\Gamma(R+ph-1)} \prod_{i=1}^{p} \left[r_i^{-r_i ph/R} \Gamma(r_i + (\frac{r_i}{R})^{ph} - 1) \right] \tag{5.38}$$

$$\text{where} \quad K = \frac{\Gamma(R-1)}{\prod_{i=1}^{p} \Gamma(r_i - 1)} \tag{5.39}$$

Applying the Inverse Mellin Transform [88] to $E(L^h)$, the pdf of L is

$$f(\ell) = K \cdot \frac{1}{2\pi i} \int_{-i\infty}^{i\infty} \ell^{-h-1} \cdot \frac{\prod_{i=1}^{p} [(k_i)^{-phk_i} \Gamma[(pm+ph+\delta)k_i - 1]]}{\Gamma(pm+\delta+ph-1)} dh \tag{5.40}$$

where

δ = adjustment or convergence factor

pm = $R - \delta$

k_i = $r_i/R; \quad k_1 + k_2 + \cdots + k_p = 1$

define

$$t = m + h \qquad ;\text{i.e.} \qquad h = t - m \tag{5.41}$$

From (5.40),(5.41), we get

$$f(\ell) = K_1 \frac{1}{2\pi i} \int_{m-i\infty}^{m+i\infty} \ell^{-t} \phi(t) dt \tag{5.42}$$

$$\text{where} \quad K_1 = K \ell^{m-1} \prod_{i=1}^{p} (k_i)^{-pmk_i} \tag{5.43}$$

$$\text{and} \quad \phi(t) = \frac{\prod_{i=1}^{p} \left[(k_i)^{-ptk_i} \Gamma((pt + \delta)k_i - 1) \right]}{\Gamma(pt + \delta - 1)} \tag{5.44}$$

Using the asymptotic expansion for the logarithm of the Gamma function [5:pp.312] then–

$$\phi(t) = K_2 \cdot t^{-v} \left[1 + \frac{q_1}{t} + \frac{q_2}{t^2} + \cdots \right] \tag{5.45}$$

$$\text{where} \quad K_2 = (2\pi)^{\left(\frac{p-1}{2}\right)} p^{-v} \prod_{i}^{p} k_i^{\left(\delta k_i - \frac{3}{2}\right)} \tag{5.46}$$

$$\text{and} \quad v = 3(p-1)/2 \tag{5.47}$$

The coefficients q_r are recursively determined using the following relations

$$q_r = \sum_{k=1}^{r} k A_k q_{r-k}/r \qquad , q_0 = 1 \tag{5.48}$$

$$A_r = \frac{(-1)^r}{r(r+1)p^r} \left[B_{r+1}(\delta - 1) - \sum_{i}^{p} \frac{B_{r+1}(\delta k_i - 1)}{k_i^r} \right] \tag{5.49}$$

Eq (5.45) shows that

$$\phi(t)/K_2 = O(t^{-v}) \tag{5.50}$$

with real part of t tending to ∞; $\phi(t)$ has therefore the following exact representation as a factorial series [72, 74]

$$\phi(t) = K_2 \sum_{i=0}^{\infty} R_i \frac{\Gamma(t+a)}{\Gamma(t+a+v+i)} \qquad , R_0 = 1 \tag{5.51}$$

where a is a convergence factor chosen such that $R_1 = 0$, and the R_i's are obtained using (5.32) Using (5.51) in (5.42), the *pdf* of L is [71]

$$f(\ell) = K_3 \cdot \sum_{i=0}^{\infty} R_i \frac{\ell^{m+a-1}(1-\ell)^{v+i-1}}{\Gamma(v+i)} \tag{5.52}$$

$$\text{where} \qquad K_3 = (2\pi)^{\frac{(p-1)}{2}}(p)^{-v}\Gamma(R-1)\prod_i^p \left[k_i^{(Rk_i-\frac{1}{2})}/\Gamma(Rk_i-1)\right] \tag{5.53}$$

In Eq (5.49), choose δ such that $A_1 = 0$, then

$$\delta = \frac{13}{18}[\sum_i^p (k_i)^{-1} - 1]/(p-1) \tag{5.54}$$

In Eq (5.51), choose a such that $R_1 = 0$, then

$$a = (1.0 - v)/2.0 \tag{5.55}$$

From (5.52), the cdf of L is

$$\begin{aligned} F(\ell) &= P(L \le \ell) = \int_0^{\ell} f(\ell)d\ell \\ &= K_3 \sum_{i=0}^{\infty} R_i \int_0^{\ell} \frac{\ell^{m+a-1}(1-\ell)^{v+i-1}}{\Gamma(v+i)} d\ell \\ &= K_3 \sum_{i=0}^{\infty} R_i' \cdot I_\ell(m+a, v+i) \qquad (5.56) \\ \text{where} \quad R_i' &= R_i[\Gamma(m+a)/\Gamma(m+a+v+i)] \qquad (5.57) \end{aligned}$$

VI. THE ASYMPTOTIC DISTRIBUTION OF THE TEST STATISTIC

When the exact method for comparing different sets of software failure data was applied, some sets were very large and sometimes exceeded eight hundred failure times. The calculations for these large sets were very time consuming, which made it necessary to think of another efficient method for calculating the cumulative density function (cdf) with accuracy approaching that of the exact method. In this chapter, a useful asymptotic expansion of the distribution is obtained up to the order of R^{-3}, where R is the total number of failures. The second term in this expansion is of the order of R^{-2}, and so can be used to obtain accurate approximations to the percentage points of the test statistic. In fact, the first term alone, which is a single beta distribution, provides a powerful approximation even for moderately large values of R. Of course, for a small number of total failures (less than 20 failures), one has to use the exact distribution obtained in the previous chapter.

6.1 Preliminaries

From the previous chapter, the exact cdf $F(\ell)$ is

$$F(\ell) = K_3 \cdot \sum_{i=0}^{\infty} R_i \frac{\Gamma(m+a)}{\Gamma(m+a+v+i)} I_\ell(m+a, v+i) \tag{6.1}$$

$$\text{where} \qquad K_3 = (2\pi)^{\frac{(p-1)}{2}} (p)^{-v} \Gamma(R-1) \prod_{i=1}^{p} \left[k_i^{(Rk_i - \frac{3}{2})} / \Gamma(Rk_i - 1) \right] \tag{6.2}$$

Knowing that $pm = R - \delta$, K_3 can be expressed as

$$\begin{aligned} K_3 &= \frac{\Gamma(pm+\delta-1)}{\prod_{i=1}^{p} \Gamma((pm+\delta)k_i - 1)} p^{-v} \prod_{i=1}^{p} (k_i)^{pmk_i + \delta k_i - \frac{3}{2}} (2\pi)^{(\frac{p-1}{2})} \\ &= K(m) \cdot p^{-v} \prod_{i=1}^{p} (k_i)^{pmk_i + \delta k_i - \frac{3}{2}} (2\pi)^{(\frac{p-1}{2})} \qquad (6.3) \\ \text{where} \quad K(m) &= \frac{\Gamma(pm+\delta-1)}{\prod_{i=1}^{p} \Gamma(pmk_i + \delta k_i - 1)} \qquad (6.4) \end{aligned}$$

The following lemma will help in simplifying the derivation of the asymptotic distribution.

Lemma VI.1 *The following expansion for the ratio of two Gamma functions holds [69:359]*

$$\frac{\Gamma(m+\delta+a)}{\Gamma(m+\delta+a+v+i)} = m^{-(v+i)}\left[1+\frac{c_{i,1}}{m}+\frac{c_{i,2}}{m^2}+\cdots\right] \quad (6.5)$$

$$\text{where} \quad c_{i,r} = \frac{(-1)^{r-1}}{r(r+1)}[B_{r+1}(a)-B_{r+1}(a+v+i)] \quad (6.6)$$

Taking the natural logarithm of $K(m)$ given in (6.4), we get

$$\begin{aligned}
\ell n K(m) &= \ell n\Gamma(pm+\delta-1)-\sum_{i=1}^{p}\ell n\Gamma(pmk_i+\delta k_i-1)\\
&= [\frac{1}{2}\ell n(2\pi)+(pm+\delta-1-\frac{1}{2})\ell n(pm)-(pm)\\
&\quad -\sum_{r=1}^{m}\frac{(-1)^r B_{r+1}(\delta-1)}{r(r+1)(pm)^r}+R_{m+1}(pm)]\\
&\quad -\sum_{i=1}^{p}[\frac{1}{2}\ell n(2\pi)+(pmk_i+\delta k_i-1-\frac{1}{2})\ell n(pmk_i)-(pmk_i)\\
&\quad -\sum_{r=1}^{m}\frac{(-1)^r B_{r+1}(\delta k_i-1)}{r(r+1)(pmk_i)^r}+R_{m+1}(pmk_i)]\\
&= \frac{1}{2}\ell n(2\pi)+pm\ell n(pm)+\delta\ell n(pm)-\frac{3}{2}\ell n(pm)-(pm)\\
&\quad -\sum_{r=1}^{m}\frac{(-1)^r B_{r+1}(\delta-1)}{r(r+1)(pm)^r}+R_{m+1}(pm)-\frac{p}{2}\ell n(2\pi)\\
&\quad -\sum_{i=1}^{p}(pmk_i+\delta k_i-\frac{3}{2})\ell n(pm)-\sum_{i=1}^{p}(pmk_i+\delta k_i-\frac{3}{2})\ell n k_i\\
&\quad +\sum_{i=1}^{p}(pmk_i)+\sum_{i=1}^{p}\sum_{r=1}^{m}\frac{(-1)^r B_{r+1}(\delta k_i-1)}{r(r+1)(pmk_i)^r}+R_{m+1}(pmk_i)\\
&= \frac{(1-p)}{2}\ell n(2\pi)+pm\ell n(pm)+\delta\ell n(pm)-\frac{3}{2}\ell n(pm)-(pm)\\
&\quad -\sum_{r=1}^{m}\frac{(-1)^r B_{r+1}(\delta-1)}{r(r+1)(pm)^r}+R_{m+1}(pm)-pm\ell n(pm)-\delta\ell n(pm)\\
&\quad +\frac{3p}{2}\ell n(pm)-\sum_{i=1}^{p}[(pmk_i+\delta k_i-\frac{3}{2})\ell n k_i+pm\\
&\quad +\sum_{i=1}^{p}\sum_{r=1}^{m}\frac{(-1)^r B_{r+1}(\delta k_i-1)}{r(r+1)(pmk_i)^r}+R_{m+1}(pmk_i)
\end{aligned}$$

$$i.e. \quad \ell n K(m) = \frac{(1-p)}{2}\ell n(2\pi) + \frac{3}{2}(p-1)\ell n(pm) - \sum_{i=1}^{p} \ell n k_i^{(pmk_i+\delta k_i-\frac{3}{2})} + \sum_{r=1}^{m}\frac{a_r}{m^r} + O(m^{r-1}) \tag{6.7}$$

$$\text{where} \quad a_r = \frac{(-1)^r}{r(r+1)p^r}\left[\sum_{i=1}^{p}\frac{B_{r+1}(\delta k_i - 1)}{k_i^r} - B_{r+1}(\delta - 1)\right] = -A_r \tag{6.8}$$

$\ell n K(m)$ can be expressed as

$$\ell n K(m) = \ell n\left[(2\pi)^{\frac{(1-p)}{2}}(pm)^{\frac{3}{2}(p-1)}\prod_{i=1}^{p} k_i^{-(pmk_i+\delta k_i-\frac{3}{2})} e^{\sum_{r=1}^{m}\frac{a_r}{m^r}}\right] \tag{6.9}$$

$$\begin{aligned} \text{so} \quad K(m) &= (2\pi)^{\frac{(1-p)}{2}}(pm)^{\frac{3}{2}(p-1)}\prod_{i=1}^{p} k_i^{-(pmk_i+\delta k_i-\frac{3}{2})} e^{\sum_{r=1}^{m}\frac{a_r}{m^r}} \\ &= (2\pi)^{\frac{(1-p)}{2}}(pm)^{\frac{3}{2}(p-1)}\prod_{i=1}^{p} k_i^{-(pmk_i+\delta k_i-\frac{3}{2})}\left[1 + \frac{T_1}{m} + \frac{T_2}{m^2} + \cdots\right] \end{aligned} \tag{6.10}$$

The coefficients T_r are recursively determined from the following relation

$$T_r = \frac{1}{r}\sum_{k=1}^{r} k a_k T_{r-k} \qquad , T_0 = 1 \tag{6.11}$$

Substituting in (6.3), we get

$$\begin{aligned} K_3 &= (2\pi)^{\frac{(1-p)}{2}}(pm)^{v}\prod_{i=1}^{p} k_i^{-(pmk_i+\delta k_i-\frac{3}{2})}\left[1 + \frac{T_1}{m} + \frac{T_2}{m^2} + \cdots\right] \\ &\quad \cdot p^{-v}\prod_{i=1}^{p}(k_i)^{pmk_i+\delta k_i-\frac{3}{2}}(2\pi)^{(\frac{p-1}{2})} \\ &= m^{v}\left[1 + \frac{T_1}{m} + \frac{T_2}{m^2} + \cdots\right] \end{aligned} \tag{6.12}$$

(6.13)

$$\text{where} \quad v = \frac{3}{2}(p-1) \tag{6.14}$$

Choosing δ such that $T_1 = 0$, makes $a_1 = 0$ and $T_2 = a_2$, then substituting in (6.1), we get

$$F(\ell) = m^v \left[1 + \frac{T_2}{m^2} + \frac{T_3}{m^3} \cdots\right] \cdot \sum_{i=0}^{\infty} R_i \frac{\Gamma(m+a)}{\Gamma(m+a+v+i)} I_\ell(m+a, v+i) \tag{6.15}$$

Using lemma (VI.1) for $\dfrac{\Gamma(m+a)}{\Gamma(m+a+v+i)}$, $F(\ell)$ will be

$$\begin{aligned}
F(\ell) &= m^v \left[1 + \frac{T_2}{m^2} + \frac{T_3}{m^3} + \cdots\right] \sum_{i=0}^{\infty} R_i m^{-(v+i)} \left[1 + \frac{c_{i,1}}{m} + \frac{c_{i,2}}{m^2} + \cdots\right] \\
&\quad \cdot I_\ell(m+a, v+i) \\
&= \left[1 + \frac{T_2}{m^2} + \frac{T_3}{m^3} + \cdots\right] \sum_{i=0}^{\infty} R_i \left[\frac{1}{m^i} + \frac{c_{i,1}}{m^{i+1}} + \frac{c_{i,2}}{m^{i+2}} + \cdots\right] \\
&\quad \cdot I_\ell(m+a, v+i) \\
&= [1 + \frac{T_2}{m^2} + \frac{T_3}{m^3} \cdots][(R_0 + \frac{c_{0,1}}{m} + \frac{c_{0,2}}{m^2} + \cdots) I_\ell(m+a, v) \\
&\quad + R_1(\frac{1}{m} + \frac{c_{1,1}}{m^2} + \cdots) I_\ell(m+a, v+1) \\
&\quad + R_2(\frac{1}{m^2} + \frac{c_{2,1}}{m^3} + \cdots) I_\ell(m+a, v+2) \\
&= R_0 I_\ell(m+a, v) + \frac{1}{m}\left[c_{0,1} I_\ell(m+a, v) + R_1 I_\ell(m+a, v+1)\right] \\
&\quad + \frac{1}{m^2}[c_{0,2} I_\ell(m+a, v) + R_1 c_{1,1} I_\ell(m+a, v+1) \\
&\quad + R_2 I_\ell(m+a, v+2) + T_2 R_0 I_\ell(m+a, v)] + O(m^{-3})
\end{aligned} \tag{6.16}$$

Since $R_0 = 1$, a is chosen such that $R_1 = 0$, and δ is chosen such that

$T_1 = a_1 = 0$, then from the recurrence relations $c_{0,1} = 0$, and $F(\ell)$ will be

$$\begin{aligned}
F(\ell) &= I_\ell(m+a, v) + \frac{1}{m^2}[c_{0,2} I_\ell(m+a, v) + R_2 I_\ell(m+a, v+2) \\
&\quad + T_2 I_\ell(m+a, v)] + O(m^3) \\
&= I_\ell(m+a, v) + \frac{1}{m^2}[(c_{0,2} + T_2) I_\ell(m+a, v) + R_2 I_\ell(m+a, v+2)] \\
&\quad + O(m^3)
\end{aligned} \tag{6.17}$$

Since $R_2 = -T_2 - c_{0,2}$, then the asymptotic distribution of the test statistic is

$$F(\ell) = I_\ell(m+a,v) + \frac{R_2}{m^2}[I_\ell(m+a,v+2) - I_\ell(m+a,v)] + O(m^3) \tag{6.18}$$

In practice, it is found that this asymptotic distribution is good when the number of failures in the set is greater than twenty.

VII. DESIGN AND APPLICATION OF THE PMS ENVIRONMENT

In the preceding two chapters, the mathematical solution for the proposed method for solving the problem of proper model selection has been developed. This is not the end in itself. This mathematical solution should be transformed into a useful algorithm for the practical application of this mathematical solution. The algorithm is then coded, using the appropriate programming language, into an environment. In this way, the solution can even be modified to solve other similar problems that may even exist in other fields. The transformation of a mathematical solution of a problem into an environment is the link between two sides. The mathematical solution of the problem, the design of an environment, and coding of this design, using an appropriate programming language, are one side. The other side is the user who will use the environment, and provide the feed back about the tool to the designer and the programmer.

7.1 Definition of the Proper Model Selection (PMS) Environment

The first question to be asked is what the Proper Model Selection (PMS) environment can do and how. As the name indicates, it helps in selecting the proper software reliability model for a given set of data. This environment is very simple and user-friendly. The user will keep in mind the accuracy of calculations needed (exact or asymptotic), depending on the computer time available and the size of the failure data on hand (asymptotic method is preferable for failure sets more than 25 in size). All that the user needs to know after that is whether the data he or she has, are in the form of failure times or times between failures and the name of the file where the data are stored. The environment will take these data and estimate the sample size (which is not known in case of software failures). The maximum likelihood method is used to estimate the sample size as given in Appendix B. After estimating the sample size, the environment will check whether the given set of data can be from an exponential distribution, which is the most common distribution for software reliability studies. The method used for testing the exponentiality of software failure sets is the

one referred to as the G *test*. This method is based on the so-called Gini statistic and is discussed by Gail and Gastwirth [29]. The test has good power against some other alternatives [47:446] specially with type *II* censored data. If the data did not provide evidence against exponentiality, then the PMS will calculate the likelihood ratio test criterion for the given set and compare it with the percentage points at different significance levels to determine if the given set of data can be considered equal to one or more of the software failure sets given by Musa [64] (refered to as DACS failure sets), or by Littlewood [52]. A table will be produced by the environment, telling whether the set under test is equal to one or more of the other sets, and the maximum significance level at which they can be considered equal. If the given set of data provide evidence against exponentiality, then extra data may be needed.

Along with the PMS, a table generation environment is developed to give the percentile points at different significance levels for different failure sizes, and at different numbers of samples.

7.2 *Design of the PMS*

The Function-Oriented Design method has been used for the design of the environment as shown in Fig (7.1). This method makes use of the data flow diagrams and the structure charts for the environment. The Proper Model Selection environment (PMS), is designed to work with Personal Computers using *MS DOS* operating system. This will widen the base of users of the environment and will enhance the tool itself, by adding more data files to existing ones. The PMS can be easily modified to incorporate new mathematical methods, or to improve those already in use. For the implementation of the design, the C language is chosen as the programming language. Code listings for implementing the design of the PMS environment shown in Fig (7.1) are given in Appendix C.

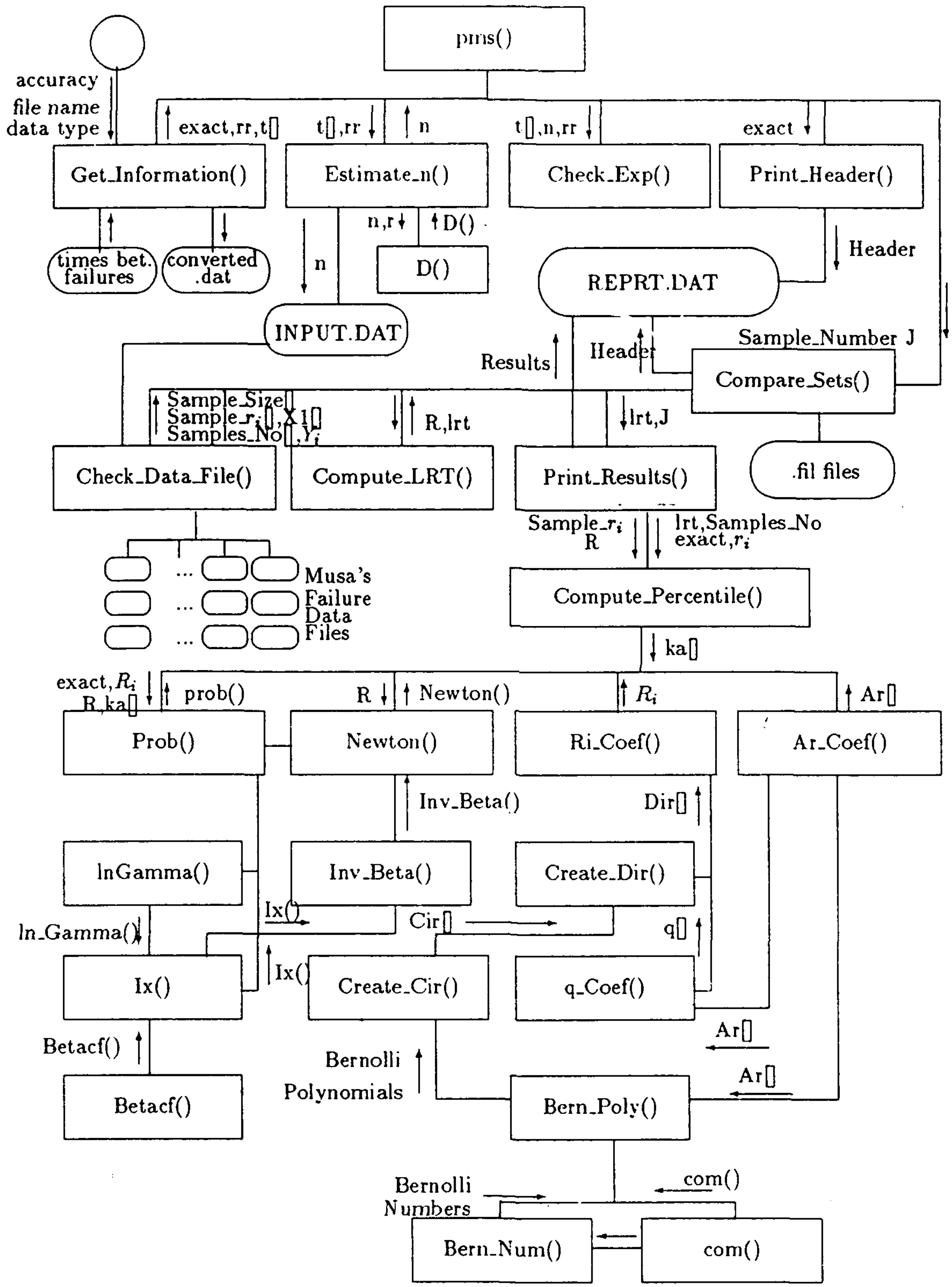

Figure 7.1. A Structure Chart for the PMS Environment

7.3 A Practical Example for Using the PMS Environment

To demonstrate how to use the PMS environment, we have chosen Set 2 of the DACS software failure data supplied by Musa [64] as an example for the demonstration. This set of data is stored in the form of times between failures as given in table (7.1).

191	50	6900	2519
222	660	3300	6890
280	1507	1510	3348
290	625	195	2750
290	912	1956	6675
385	638	135	6945
570	293	661	7899
610	1212	50	
365	612	729	
390	675	900	
275	1215	180	
360	2715	4225	
800	3551	15600	
1210	800	300	
407	3910	9021	

Table 7.1. Set 2 of DACS failure data in [64]

Assuming that the set is stored in a file "set2.dat", then the sequence for running the environment will start by typing the executable file name "pms" and continues as follows :

```
Enter Type of Calculations Exact (e|E) OR Asymptotic (a|A)?

a

asymptotic calculations requested



OK ! Now Enter The Name of Data File ?


set2.dat
```

```
Enter the Type of the Data

Failures Times (f|F) OR Time Between Failures (t|T)?


f

failure times


        Supplied Data Provided Evidence Towards Exponentiality


                   Likelihood Ratio Test Program

           Asymptotic Method for Checking Equality of Data

                  and Proper  Model Selection

           The Symbols in the table are Y(es) or N(o)

              The Value Under Yes or No is (L-C)

             For Equality L must be greater than C


then Table D-2 in APPENDIX D will be generated.
```

from the table the following information can be extracted:

1. Set 2 of DACS can be considered equal to both of set 1 of DACS sets [64] and set 2 of Littlewood sets [52].
2. It is also clear that set 2 is closer to the former set than the latter one ($L-C$=0.035 vs 0.017 at significance level =0.05)
3. The set can also be considered to be equal to set 17 of DACS sets, but at a lower significance level (0.005).

The question to be answered next is:

Which Software Reliability Model is the Most Appropriate One to be Used with This Set of Data ?

To answer this question, it is necessary to find out the model or models that fitted properly in the past with those sets of data that proved to be equal to the set under investigation (i.e. set 1 of DACS sets and set 2 of Littlewood sets). For the first set, full Analyses were performed by Littlewood [52:184] on this set of data, using the Median Variability, the Rate Variability, the Braun Statistic, the $u-$plot, the $y-$plot, and the Prequential Likelihood analysis. The results of these analyses strongly suggested that the Littlewood Model (L) [49] and the Littlewood Non Homogeneous Poission Process Model (LNHPP) [1:955] are the most appropriate ones. This means that these two models are also the best candidates to be used with the set of data under investigation. On the other hand, one can also use the software reliability model or models that proved to be the best with the set 2 of Littlewood data (despite the fact that it is less closer than the previous one, as explained before). *Similar analysis done by Abdel-Ghaly and Others* [1] suggested that the Littlewood Model [49], the Littlewood-Verrall Model (LV) [55], or the Keiler-Littlewood Model (KL) [41, 42] are the best models to fit with the second set of Littlewood data, and accordingly, with the set of data under test. From the above discussion, it is clear also that the Littlewood Model [49] fitted both of the two sets that proved to be equal to the data set under consideration. This strongly suggests that the Littlewood Model [49] is the best one to be used with the set of data under study.

Full Analyses of DACS failure data and the two sets of data supplied by Littlewood [52] are given in APPENDIX D.

VIII. SUMMARY AND RECOMMENDATIONS FOR FUTURE WORK

A large number (currently about 100 [26:322]) of software reliability models have been developed within the last two decades. Unfortunately, each of these models can produce a very different answer from the others when applied to predict the reliability of a software system. Further, a model that seems to be the best for one set of data, may give very poor results with other sets. This is not surprising, since the assumptions for each model cannot be satisfied in all situations, ' to the variations in the development and testing environments for each project. The above problems, as well as time and cost constraints, made many software practitioners avoid incorporating reliability modeling as a management technique. This situation makes it necessary to devote less effort for developing more software reliability models, and concentrate on finding tools for selecting the proper model for a given set of data.

The Proper Model Selection (PMS) environment is one approach for solving this problem. It uses the same model that proved to be the best for an equal set of data in the past. It performs a test of equality of a given set of software failures with each of the other sets that previously fitted with different models, to determine how close the set under test is to each of other sets. if past predictions of a software reliability model have been in close accord with actual behavior for a particular data set, then it will be reasonable to use the same model with an equal set of data. The PMS can help in selecting the proper reliability model without evaluating different models, comparing them, and then choosing the best one. It can be simply stated as "*matching a software reliability model to a given set of failure data*".

The (PMS) is a simple and objective method for selecting a proper model for a given set of data. The main advantage of this method is that one will make use of the previous effort consumed, when different models were applied with different sets of data, and when different techniques were applied, to measure the predictive quality of software models with certain sets of data. The PMS environment will provide a strong framework, and a useful data base system, which will result in the

user confidence of the software reliability calculations he/she makes. This will help in establishing a software reliability theory similar to that of hardware.

Based on the assumptions stated initially, and the observations found during the development and the application of the PMS environment, the following recommendations are proposed for future work:

1. Despite the fact that estimating the sample size using the maximum likelihood method is straightforward, some failure sets showed that the size is very close to the number of collected failures. Other methods for dealing with such cases, need to be added to the environment.

2. Two sets of failure data (SETS 27,SS4 of DACS failure sets) were not equal to any other sets. This may be due to the limited number of sets of software failure data used for comparison, or more data may need to be collected. The former case can be solved by adding more failure sets to the environment.

3. The studies for deciding which model fits with which set of data are still limited [41, 42, 52]. More studies of this kind are needed. This will make it possible to modify the environment in the future to provide the best candidate models to be used along with each equal set.

Appendix A. *Calculations of the Adjustment Factors*

In this Appendix, the values of the convergence factors a and δ used in chapters 5 and 6. The values of these two factors are calculated in the following two sections

A.1 Calculation of δ

δ is an adjustment factor defined such that

$$A_1 = 0; \tag{A.1}$$

where

$$A_r = \frac{(-1)^r}{r(r+1)p^r}\left[B_{r+1}(\delta - 1) - \sum_{i=1}^{p}\frac{B_{r+1}(\delta k_i - 1)}{k_i^r}\right] \tag{A.2}$$

The symbols used in the equation are

p	the number of samples
k_i	the percentage of failures in sample i
$B_r(\cdot)$	Bernoulli Polynomial of degree r and order unity

taking $r = 1$, we get

$$A_1 = \frac{-1}{2p}\left[B_2(\delta - 1) - \sum_{i=1}^{p}\frac{B_{r+1}(\delta k_i - 1)}{k_i}\right] \tag{A.3}$$

Knowing that the second Bernoulli Polynomial $B_2(h)$ is

$$B_2(h) = h^2 + h + \frac{1}{6} \tag{A.4}$$

Substituting in (A.3), we have

$$
\begin{aligned}
A_1 &= \frac{-1}{2p}\left[((\delta-1)^2-(\delta-1)+\frac{1}{6})-\sum_{i=1}^{p}\frac{(\delta k_i-1)^2-(\delta k_i-1)+\frac{1}{6}}{k_i}\right] \\
&= \frac{-1}{2p}\left[(\delta^2-3\delta+\frac{13}{6})-\sum_{i=1}^{p}\frac{(\delta^2k_i^2-3\delta k_i+\frac{13}{6})}{k_i}\right] \\
&= \frac{-1}{2p}\left[(\delta^2-3\delta+\frac{13}{6})-\sum_{i=1}^{p}(\delta^2 k_i-3\delta+\frac{13}{6}(k_i^{-1}))\right] \\
&= \frac{1}{2p}\left[-\delta^2+3\delta-\frac{13}{6}+\delta^2-3\delta p+\frac{13}{6}\sum_{i=1}^{p}(k_i^{-1})\right] \\
&= \frac{1}{2p}\left[3\delta(1-p)+\frac{13}{6}[\sum_{i=1}^{p}(k_i^{-1})-1]\right] \qquad (A.5)
\end{aligned}
$$

To get δ, we put $A_1 = 0$ so

$$
\begin{aligned}
3\delta(p-1) &= \frac{13}{6}[\sum_{i=1}^{p}(k_i^{-1})-1] \qquad \text{i.e.} \\
\delta &= \frac{13}{18}[\sum_{i=1}^{p}(k_i^{-1})-1]/3(p-1) \qquad (A.6)
\end{aligned}
$$

A.2 Calculation of a

a is a convergence factor chosen such that $R_1 = 0$, and the coefficients R_i are obtained using the following recurrence relations from [71]

$$
\begin{aligned}
\sum_{j=0}^{i} R_{i-j}d_{i-j,j} &= q_i(i=1,2,3,\cdots,R_0=1) \qquad (A.7) \\
d_{i,r} &= \sum_{k=1}^{r} kc_{i,k}d_{i,r-k}/r, \qquad d_{i,0=1} \qquad (A.8) \\
c_{i,r} &= \frac{(-1)^{r-1}}{r(r+1)}[B_{r+1}(a)-B_{r+1}(a+v+i)] \qquad (A.9)
\end{aligned}
$$

where $v = 3(p-1)/2$ and q_r is determined using the following relation

$$q_r = \sum_{k=1}^{r} k A_k q_{r-k}/r \qquad , q_0 = 1 \tag{A.10}$$

the value of R_1 can be found from

$$R_1 d_{1,0} + R_0 d_{0,1} = q_1 = A_1 \tag{A.11}$$

where $A_1 = 0$ as discussed above. If $R_1 = 0$ is desired, then

$$d_{0,1} = c_{0,1} d_{0,0} = 0 \tag{A.12}$$

this means $c_{0,1} = 0$ i.e.

$$\frac{1}{2}[B_2(a) - B_2(a+v+i)] = 0 \tag{A.13}$$

Substituting for the Bernoulli Polynomials, we get

$$\frac{1}{2}[a^2 - a + \frac{1}{6} - (a+1)^2 + (a+1) - \frac{1}{6}] = 0 \tag{A.14}$$

which reduces to

$$v^2 + 2av - v = 0 \tag{A.15}$$

solving this equation gives

$$a = \frac{1-v}{2} \tag{A.16}$$

Appendix B. *Maximum Likelihood Estimate of Sample Size n*

Let $X_{(1)} \leq X_{(2)} \leq ... \leq X_{(r)}$ be the order statistic from the 2-parameter exponential distribution, then the likelihood function for a random sample of size n is

$$\begin{aligned} L(x_{(1)}, x_{(2)}, ..., x_{(r)}, \sigma, \theta) &= \frac{n!}{(n-r)!}\frac{1}{\sigma^r}\exp[\frac{1}{\sigma}(\sum_{i=1}^{r}(x_{(i)}-\theta)+(n-r)(x_{(r)}-\theta))] \\ &= \frac{n!}{(n-r)!}\frac{1}{\sigma^r}\exp[-\frac{1}{\sigma}(x_{(r)}-\theta)(\sum_{i=1}^{r}\frac{r(x_{(i)}-\theta)}{r(x_{(r)}-\theta)}+(n-r))] \\ &= \frac{n!}{(n-r)!}\frac{1}{\sigma^r}\exp[-\frac{1}{\sigma}(x_{(r)}-\theta)(rs+(n-r))] \qquad (B.1) \end{aligned}$$

where $s = \sum_{i=1}^{r} \frac{(x_{(i)}-\theta)}{r(x_{(r)}-\theta)}$ is the observed value of the random variable

$$S = \sum_{i=1}^{r} \frac{(X_{(i)}-\theta)}{r(X_{(r)}-\theta)} \qquad (B.2)$$

Noting that for each n ,we've found in chapter 2 that equation (B.1) was maximized by having

$$\begin{aligned} \hat{\sigma} &= \frac{1}{r}\left[\sum_{i=1}^{r} x_{(i)} + (n-r)x_{(r)} - nx_{(1)}\right] \qquad (B.3) \\ \text{and } \hat{\theta} &= x_{(1)} \qquad (B.4) \end{aligned}$$

Denote (B.1) with $\hat{\sigma}$ and $\hat{\theta}$ substituted by $L(x \mid n)$. Let $\mathcal{L}(x \mid n)$ be $\log L(x \mid n)$, then (B.1) is maximized by the n satisfying

$$\mathcal{L}(x \mid n+1) - \mathcal{L}(x \mid n) \leq 0 \leq \mathcal{L}(x \mid n) - L(x \mid n-1) \qquad (B.5)$$

Following the same procedure as Blumenthal [12], equation (B.5) is achieved by $\hat{n}$ such that

$$D(\hat{n}) \leq s < D(\hat{n}+1) \qquad (B.6)$$

where

$$D(\hat{n}) = \left\{r(1 - [1 - (r/\hat{n})]^{(1/r)})\right\}^{-1} + 1 - (\hat{n}/r) \tag{B.7}$$

If there is an upper bound m on the possible values of n, then the estimator would be

$$\hat{n}_m = \min(m, \hat{n}) \tag{B.8}$$

Appendix C. *Code Listings*

```
/*******************************************************
** DATE: 01/16/1992                                  **
** VERSION: 1.0                                      **
** TITLE: Main Program File                          **
** FILENAME: pms.c                                   **
** COORDINATOR: Salah Amin Elewa                     **
** PROJECT:Development of an Environment for Software **
**                 Reliability Model Selection.      **
** OPERATING SYSTEM: MS DOS version 2.0 or higher    **
** LANGUAGE: Turbo C (2.1)                           **
** FILE PROCESSING: Compile and link with files prep.c**
**     compute.c, coefs.c, and  prob.c,              **
** CONTENTS: 1.0 main () - executive module          **
** FUNCTION: This file contains  the main program of  **
**     the project                                    **
*******************************************************/
#include <conio.h>
#include "defs.h"

extern void     Get_Information(void);
extern void     Estimate_n(void);
extern void     Check_Exp(void);
extern void     Print_Table_Header(void);
extern void     Compare_Sets(int);
```

```
/*******************************************************
** DATE: 01/16/1991                                   **
** VERSION: 1.0                                       **
** MODULE NAME: main                                  **
** MODULE NUMBER: 1.0                                 **
** DESCRIPTION: This main program in the environments **
** PASSED VARIABLES: None                             **
** RETURNS: None                                      **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT:None                               **
** MODULES CALLED: Get_Information()                  **
**                 Estimate_n()                       **
**                 Check_Exp()                        **
**                 Compare_Sets()                     **
** CALLING MODULES: None                              **
** AUTHOR: Salah Amin Elewa                           **
** HISTORY:                                           **
**       1.0 Salah Amin Elewa 01/16/1992              **
**           original version                         **
*******************************************************/
void main()
{
  int     K;

  clrscr();
  Get_Information();
  Estimate_n();
  Check_Exp();
  Print_Table_Header();
  for (K = 1; K < DATA_FILES; K++)
    Compare_Sets(K);
  return;
}
```

```
/*******************************************************
** DATE:01/25/1992                                    **
** VERSION: 1.0                                       **
** TITLE: Failure Data Processor                      **
** FILENAME: Prep.c                                   **
** COORDINATOR: Salah Amin Elewa                      **
** PROJECT:Development of an Environment for Software **
**                 Reliability Model Selection.       **
** OPERATING SYSTEM: MS DOS version 2.3 or higher     **
** LANGUAGE: Turbo C  (2.1)                           **
** FILE PROCESSING: Compile and link with files pms.c **
**     compute.c, coefs.c, and  prob.c.               **
** CONTENTS:                                          **
**            1.1.1 - Get_Information()               **
**            1.1.2 - Estimate_n()                    **
**            1.1.3 - Check_Exp()                     **
**            1.1.4 - D()                             **
**            1.1.5 - Print_Table_Header()            **
** FUNCTION: The modules in this file get information **
**      about calculation method, how data are stored **
**      (failure times or times between failures),    **
**      estimate sample size, check exponentiality of **
**      failure data and print table header.          **
*******************************************************/
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#include "defs.h"
extern boolean  exact;

void   Get_Information(void);
void   Print_Table_Header(void);
```

```
void   Estimate_n(void);
void   Check_Exp(void);
void   Print_Table_Header(void);
double D(int, int);

static int     n, rr;
static double  t[MAX_FAILURES];
static FILE    *fp100;
FILE           *fp5;
```

```
/*******************************************************
** DATE: 01/25/1992                                   **
** VERSION: 1.0                                       **
** MODULE NAME: Get_Information()                     **
** MODULE NUMBER: 1.1.1                               **
** DESCRIPTION: This module is used for obtaining inf-**
**       ormation about the required accuracy of calcu-*
**       lations, the name of the file containing     **
**       failure data and the form of failure data. It**
**       stores failure data as failure times if it was*
**       stored as time between failures.             **
** PASSED VARIABLES: None                             **
** RETURNS:None                                       **
** GLOBAL VARIABLES USED: exact, t[],rr               **
** GLOBAL VARIABLES CHANGED: exact, t[],rr            **
** FILES READ: The file containing failure data       **
** FILES WRITTEN:"converted.dat" a file with failure  **
**      data stored in the form of failure time       **
** HARDWARE INPUT: Keyboard                           **
** HARDWARE OUTPUT: Screen                            **
** MODULES CALLED: None                               **
** CALLING MODULES: The main program                  **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**       1.0 Salah A. Elewa 01/25/1992                **
**           original version                         **
*******************************************************/
void Get_Information(void)
{
  int     ch;
  char    FileName[MAX_STRING];
  FILE    *fp300;

  if ((fp300 = fopen("convertd.dat", "w+")) == NULL)
```

```
{
  puts("\t can not open converted.dat\n");
  exit(1);
}
puts("\n\n\tEnter Type of Calculations Exact (e|E) OR Asymptotic (a|A)?\n\n");
for (;;)
{
  ch = getche();
  if (ch == 'a' || ch == 'A')
  {
    puts("symptotic calculations requested\n\n");
    exact = FALSE;
  }
  else if (ch == 'e' || ch == 'E')
  {
    puts("xact calculations requested\n\n");
    exact = TRUE;
  }
  else
  {
    printf("\n\t\t\t Sorry!   %c  is Unknown Choice", ch);
    puts("\n\t\t   The Letters a,A,e,E are Acceptable Choices");
    puts("\n\t\t Try Again OR Press Ctrl+Break to Quit\n\n\n");
  }
  if (ch == 'a' || ch == 'A' || ch == 'e' || ch == 'E')
    break;
}

puts("\t\t OK ! Now Enter The Name of Data File ?\n\n");
do
{
  for (ch = getchar(); isspace(ch); ch = getchar())
    ;                              /* Null Statement */
  ungetc(ch, stdin);
```

```
    gets(FileName);
    puts("\n");
    if ((fp100 = fopen(FileName, "r")) == NULL)
    {
      printf("\n\t\t     Can NOT Find The File %s\n", FileName);
      puts("\n\t\tEnter the CORRECT File Name OR Ctrl-C to Exit\n");
    }
  }
  while ((fp100 = fopen(FileName, "r")) == NULL);

  puts("\t\t Enter the Type of the Data \n");
  puts("\tFailures Times (f|F) OR Time Between Failures (t|T)?\n\n");

  for (;;)
  {                               /* infinite loop */
    ch = getche();
    if (ch == 'f' || ch == 'F')
      puts("ailure times \n\n");
    else if (ch == 't' || ch == 'T')
    {
      puts("ime between failures \n\n");
      rr = 1;
      t[0] = 0.0;
      while (!feof(fp100))
      {
        fscanf(fp100, "%lf ", &t[rr]);
       t[rr] = t[rr] + t[rr - 1];
       fprintf(fp300, "%lf\n", t[rr]);
       rr++;
      }
      fp100 = fp300;
    }
    else
    {
```

```
      printf("\n\t\t\t Sorry!   %c  is Unknown Choice", ch);
      puts("\n\t\t The Letters f,F,b,B are Acceptable Choices");
      puts("\n\t\t Try Again OR Press Ctrl+Break to Quit\n\n\n");
    }
    if (ch == 'f' || ch == 'F' || ch == 't' || ch == 'T')
      break;
  }
  rewind(fp100);
  fclose(fp300);
  return;
}
```

```
/*******************************************************
** DATE: 01/25/1992                                    **
** VERSION: 1.0                                        **
** MODULE NAME: Estimate_n()                           **
** MODULE NUMBER: 1.1.2                                **
** DESCRIPTION: This module is used to estimate the    **
**       value of the sample size n and then insert it**
**       on the top of the failure times file so that **
**       it can be read later by the environment.The  **
**       maximum likelihood estimation method is used **
**       for estimating n as described in Appendix I. **
** PASSED VARIABLES: the Pointer to the file where     **
**                   failure times are stored          **
** RETURNS: None                                       **
** GLOBAL VARIABLES USED: t[],n,rr                     **
** GLOBAL VARIABLES CHANGED: n                         **
** FILES READ: The file containing the failure times   **
**          or the times between failures              **
** FILES WRITTEN: "input.dat" file, which has the est-**
**       imated n on top and then  failure times       **
** HARDWARE INPUT: None                                **
** HARDWARE OUTPUT: None                               **
** MODULES CALLED: Module D described in Appendix I    **
** CALLING MODULES: The main program                   **
** AUTHOR: Salah A. Elewa                              **
** HISTORY:                                            **
**       1.0 Salah A. Elewa 01/25/1992                 **
**          original version                           **
*******************************************************/
void Estimate_n(void)
{
  int       i;
  double    T, max_t, SS, theta = 1.0e37;
  FILE      *fp200;
```

```
T = max_t = SS = 0.0;
if ((fp200 = fopen("input.dat", "w+")) == NULL)
{
  puts("can not open input.dat\n");
  exit(1);
}
rr = 1;
while (!feof(fp100))
{
  fscanf(fp100, "%lf ", &t[rr]);
  if (t[rr] < theta)
    theta = t[rr];
  if (t[rr] > max_t)
    max_t = t[rr];
  T = T + t[rr];
  rr++;
}
rr = rr - 1;
printf("\t\tTotal Number of Failures r= %d\n", rr);
SS =(T-(double) rr*theta)/(double) rr/(max_t-theta);
rewind(fp100);
for (i = rr;; i++)
{
  if (D(i, rr) <= SS && SS < D(i + 1.0, rr))
  {
    n = i;
    break;
  }
}
fprintf(fp200, "%d \n", n);
rewind(fp100);
while (!feof(fp100))
{
```

```
    fscanf(fp100, "%lf ", &t[rr]);
    fprintf(fp200, "%lf \n", t[rr]);
  }
  fclose(fp200);
  return;
}
```

```
/*****************************************************************
** DATE: 01/25/1992                                            **
** VERSION: 1.0                                                **
** MODULE NAME: Check_Exp()                                    **
** MODULE NUMBER: 1.1.3                                        **
** DESCRIPTION: This module is used to check whether           **
**        the failure times are exponential. It uses           **
**        G test based on the Gini statistic for check-        **
**        ing exponentiality  of type II censored data.        **
** PASSED VARIABLES: Pointer to "Input.dat" file               **
** RETURNS: None                                               **
** GLOBAL VARIABLES USED: t[],n,rr                             **
** GLOBAL VARIABLES CHANGED: None                              **
** FILES READ: "input.dat" file                                **
** FILES WRITTEN: None                                         **
** HARDWARE INPUT: None                                        **
** HARDWARE OUTPUT: Screen                                     **
** MODULES CALLED: None                                        **
** CALLING MODULES: The main program                           **
** AUTHOR: Salah A. Elewa                                      **
** HISTORY:                                                    **
**        1.0 Salah A. Elewa 01/25/1992                        **
**            original version                                 **
*****************************************************************/
void Check_Exp(void)
{
  double  time_bet[MAX_FAILURES], Wi[MAX_FAILURES];
  double  Witot, num, den;
  double  Grn, Var;
  int     i;

  Witot = num = t[0] = 0.0;
  for (i = 1; i < rr + 1; i++)
```

```
  {
    time_bet[i] = t[i] - t[i - 1];
    Wi[i] = (n - i + 1) * time_bet[i];
    Witot = Witot + Wi[i];
  }
  den = (rr - 1) * Witot;
  for (i = 1; i < rr; i++)
    num = num + i * Wi[i + 1];
  Grn = num / den;
  Var = pow(12 * (n - 1), .5) * (Grn - 0.5);

  if (Var > -2.33 && Var < 2.33)
    /* 0.01 Significance Level is Used for Calculations */
    puts("\n\t Supplied Data Provided Evidence Towards Exponentiality\n");
  else
  {
    puts("\t\t\tSupplied Data Provided Evidence Against Exponentiality\n");
    puts("\t\t\t\t Extra Data may be Needed\n");
    puts("\t\t\t\t Program will Terminate \n");
    exit(1);
  }
  rewind(fp100);
  fclose(fp100);
}
```

```
/*****************************************************
** DATE: 01/25/1992                                  **
** VERSION: 1.0                                      **
** MODULE NAME: D()                                  **
** MODULE NUMBER: 1.1.4                              **
** DESCRIPTION: This module is used for calculating  **
**       the value of the Parameter D described in   **
**       Appendix I                                  **
** PASSED VARIABLES: number of failures r and sample **
**       size n.                                     **
** RETURNS: D value                                  **
** GLOBAL VARIABLES USED: None                       **
** GLOBAL VARIABLES CHANGED: None                    **
** FILES READ: None                                  **
** FILES WRITTEN: None                               **
** HARDWARE INPUT: None                              **
** HARDWARE OUTPUT: None                             **
** MODULES CALLED: None                              **
** CALLING MODULES: Estimate_n                       **
** AUTHOR: Salah A. Elewa                            **
** HISTORY:                                          **
**       1.0 Salah A. Elewa 01/25/1992               **
**           original version                        **
*****************************************************/
double D(int n, int r)
{
  double  nn, rr, xx;

  nn = (double) n;
  rr = (double) r;
  xx = pow(rr*(1.0-(pow((1.0-(rr/nn)),(1.0/rr)))),-1.0);
  return (xx + 1.0 - (nn / rr));
}
```

```
/*******************************************************
** DATE: 01/25/1992                                   **
** VERSION: 1.0                                       **
** MODULE NAME: Print_Table_Header()                  **
** MODULE NUMBER: 1.1.5                               **
** DESCRIPTION: This module is used for printing the  **
**       table header on the screen and in the "report**
**       .dat" file.                                  **
** PASSED VARIABLES: None                             **
** RETURNS: None                                      **
** GLOBAL VARIABLES USED: exact                       **
** GLOBAL VARIABLES CHANGED: None                     **
** FILES READ: None                                   **
** FILES WRITTEN: "report.dat" file                   **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT: None                              **
** MODULES CALLED: None                               **
** CALLING MODULES: The main program                  **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**       1.0 Salah A. Elewa 01/25/1992                **
**           original version                         **
*******************************************************/
void Print_Table_Header(void)
{
  int j;

  if ((fp5 = fopen("report.dat", "w")) == NULL)
  {
    printf("can not open report.dat file\n");
    exit(1);
  }
  printf("\t\t\t Likelihood Ratio Test Program \n");
  fprintf(fp5, "\t\t\t Likelihood Ratio Test Program \n");
```

```
if (exact)
{
  printf("\t\t Exact Method for Checking Equality of Data \n");
  printf("\t\t       and Proper  Model Selection \n");
  fprintf(fp5,"\t\t Exact Method for Checking Equality of Data \n");
  fprintf(fp5,"\t\t       and Proper Model Selection\n");
}
else
{
  printf("\t\t Asymptotic Method for Checking Equality of Data\n");
  printf("\t\t            and Proper Model Selection \n");
  fprintf(fp5,"\t\t Asymptotic Method for Checking Equality of Data\n");
  fprintf(fp5,"\t\t            and Proper Model Selection\n");
}
printf("\t\t The Symbols in the table are Y(es) or N(o)\n");
fprintf(fp5,"\t\t The Symbols in the table are Y(es) or N(o)\n");
printf("\t\t     The Value Under Yes or No is (L-C)\n");
fprintf(fp5,"\t\t     The Value Under Yes or No is (L-C)\n");
printf("\t\t    For Equality L must be greater than C \n");
fprintf(fp5, "\t\t   For Equality L must be greater than C \n");

printf("┌───────┬");
fprintf(fp5, "┌───────┬");
for (j = 1; j < 8; j++)
{
  printf("───────");
  fprintf(fp5, "───────");
}

printf("───────┐\n");
fprintf(fp5, "───────┐\n");

printf("|       |                     Significance Level α   ");
printf("                    |");
```

```
fprintf(fp5,"|          |                              Significance Level  ∝  ");
fprintf(fp5,"                         |");

printf("\n|        ├");
fprintf(fp5,"\n|        ├");
for (j = 1; j < 8; j++)
{
  printf("────────┬");
  fprintf(fp5, "────────┬");
}

printf("────────┤\n");
fprintf(fp5, "────────┤\n");
printf("| F. Set|.00025 | .0005 | .001  | .0025 | .005  ");
printf("|  .01  |  .025 |  .05  |");

fprintf(fp5,"| F. Set|.00025 | .0005 | .001  | .0025 | .005  ");
fprintf(fp5,"|  .01  |  .025 |  .05  |");

return;
}
```

```
/*******************************************************
** DATE: 01/31/1992                                   **
** VERSION: 1.0                                       **
** TITLE: Computation Routines File                   **
** FILENAME: compute.c                                **
** COORDINATOR: Salah Amin Elewa                      **
** PROJECT:Development of an Environment for Software **
**                 Reliability Model Selection.       **
** OPERATING SYSTEM:MS DOS version 2.0 or higher      **
** LANGUAGE: Turbo C (2.1)                            **
** FILE PROCESSING: Compile and link with files prep.c**
**     coefs.c, prob.c, and pms.c                     **
** CONTENTS:                                          **
**          1.2.1 - Compare_Sets()                    **
**          1.2.2 - Check_Data_File()                 **
**          1.2.3 - Compute_LRT()                     **
**          1.2.4 - Compute_Percentile()              **
**          1.2.5 - Print_Results()                   **
** FUNCTION: This file contains the modules for making**
**     the comparison between two sets of data and    **
**     printing the results in the file "report.dat". **
*******************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include "defs.h"

void    Compare_Sets(int);
void    Print_Results(double, int);
void    Compute_Percentile(double, double, double *, double *);
double  Compute_LRT(void);
int     Check_Data_File(char *);
```

```
extern FILE     *fp5;
extern double   ka[];

static FILE     *fp1;
static boolean DIFFERENT = TRUE;
static double  LRT, Xi[MAX_SAMPLES], Yi[MAX_SAMPLES];
static int      R, Samples_NO,Sample_ri[MAX_SAMPLES],
       Sample_Size[MAX_SAMPLES];
```

```
/*******************************************************
** DATE: 01/31/1992                                   **
** VERSION: 1.0                                       **
** MODULE NAME: Compare_Sets()                        **
** MODULE NUMBER: 1.2.1                               **
** DESCRIPTION: This module is used to compare the two**
**       sets of data specified in the corresponding  **
**       .fil file. It first invokes Check_Data_File  **
**       routine to make sure that all required data  **
**       for comparison exist in failure times files, **
**       then it compares input file with other sets  **
**       through the loop J using the value of LRT    **
**       and the percentile point C from the routines **
**       Compute_LRT and Compute_Percentile.          **
** PASSED VARIABLES: number of set to compare with (J)**
** RETURNS: None                                      **
** GLOBAL VARIABLES USED: Total number of failures (R)**
**       ,Samples_NO, DIFFERENT,LRT                   **
** GLOBAL VARIABLES CHANGED: LRT                      **
** FILES READ: The .fil file containing two sets of   **
**       data to be compared and the files containing **
**       failure times                                **
** FILES WRITTEN: "report.dat" file                   **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT: Screen                            **
** MODULES CALLED: Check_Data_File()                  **
**                 Compute_LRT()                      **
**                 Print_Results()                    **
** CALLING MODULES: The main program                  **
** AUTHOR: Salah Amin Elewa                           **
** HISTORY:                                           **
**       1.0 Salah Amin Elewa 01/31/1992              **
**           original version                         **
*******************************************************/
```

```
void Compare_Sets(J)
{
  double   L;
  int      p = 0;
  char     fn[MAX_STRING], s[MAX_STRING];
  char     *filfile[] = {" ", "set_1.fil", "set_2.fil",
   "set_4.fil","set_5.fil","set_6.fil","set_14c.fil",
   "set_17.fil","set_27.fil","set_ss1a.fil","set_ss1c.fil",
   "set_SS3.fil", "set_SS4.fil", "Lit_2.fil", "Lit_3.fil"};
  R = Samples_NO = 0;

  if (J > 1)
    fclose(fp1);
  if ((fp1 = fopen(filfile[J], "r")) == NULL)
  {
    printf("\n can not open %s file\n", filfile[J]);
    exit(1);
  }
  fgets(s, 30, fp1); /* read 1st set name from the .fil file fp1*/
  while (!feof(fp1))
  {
    if (sscanf(s, "%s", fn))    /* assign s to file fn if exist */
    if (Check_Data_File(fn))
      p++;                        /* Check all data are found */
    fgets(s, 30, fp1);  /* read 2nd set name from the .fil file */
  }
  if (p > 1)
  {
    LRT = Compute_LRT();
    L = pow(LRT, (1.0 / (double) R));
    Print_Results(L, J);
  }
  else
  {
```

```
    printf("\n\nError LRT can not be computed for less than 2 samples\n");
    printf("\n - OR - May be one of the failure times is 0.0 \n");
    fprintf(fp5,"\nError LRT can not be computed for less than 2 samples\n");
    fprintf(fp5, "\n - OR - May be one of the failure times is 0.0 \n");
  }
  if (J == (DATA_FILES - 1))
  {
    if (!DIFFERENT)
    {
      printf("\n └───────┴─────────────────────────────────────────── ");
      printf(" ──────────────────┘ \n");
      fprintf(fp5,"\n └───────┴─────────────────────────────────────── ");
      fprintf(fp5," ──────────────────┘ \n");
    }
    else
    {
      printf("\n └──────┴───────┴──────┴──────┴──────┴──────┴── ");
      printf(" ─────┴──────┴──────┘ \n");
      fprintf(fp5, "\n └──────┴──────┴──────┴──────┴──────┴──── ");
      fprintf(fp5," ───┴──────┴──────┴──────┘ \n");
    }
    fclose(fp5);
    fclose(fp1);
  }
  return;
}
```

```
/*******************************************************
** DATE: 01/31/1992                                   **
** VERSION: 2.0                                       **
** MODULE NAME: Check_Data_File()                     **
** MODULE NUMBER: 1.2.2                               **
** DESCRIPTION: This module is used to read the files **
**      containing the failure data and return TRUE   **
**      if all information needed exist.              **
** PASSED VARIABLES: Name of file to be checked       **
** RETURNS: TRUE or FALSE (1 or 0)                    **
** GLOBAL VARIABLES USED: Sample_Size[], Sample_ri[], **
**       Samples_NO, X1[], Yi[]                       **
** GLOBAL VARIABLES CHANGED: Sample_Size[],Sample_ri[],*
**       Samples_NO, X1[], Yi[]                       **
** FILES READ: fn file passed as an argument          **
** FILES WRITTEN: None                                **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT: None                              **
** MODULES CALLED: None                               **
** CALLING MODULES: Compare_Sets()                    **
** AUTHOR: Salah Amin Elewa                           **
** HISTORY:                                           **
**        1.0 Capt. K. N. Cole 5/8 /1985              **
**            original  version                       **
**        1.1 Capt. K. N. Cole 10/25 /1986            **
**            modified for SAE system                 **
**        2.0 Salah Amin Elewa      01/31/1992        **
**            - header modified                       **
**            - names for module and some variables   **
**              changed                               **
**            - program modified to work with 2-para-**
**              meter exponential distribution and to**
**              read sample size from file "input.dat"*
*******************************************************/
```

```
int Check_Data_File(char *fn)
{
  FILE       *fp;
  boolean    error;
  char       *c;
  int        ni, ri;
  double     t, Ti, maximum_t, minimum_t;

  t = Ti = maximum_t = minimum_t = 0.0;
  if (Samples_NO >= MAX_SAMPLES)
    error = TRUE;
  else
  {
    error = FALSE;
    maximum_t = 0.0;
    c = fn;
    minimum_t = 1.0e37;
    while (isspace(*c))
      c++;
    if ((fp = fopen(c, "r")) != NULL)
    {
      fscanf(fp, " %d ", &ni);
      ri = 0;
      Ti = 0.0;
      while (!feof(fp))
      {
       t = 0.0;
       fscanf(fp, "%lf ", &t);
       if (t != 0.0)
       {
         if (t < minimum_t)
           minimum_t = t;
         if (t > maximum_t)
           maximum_t = t;
```

```
        Ti = Ti + t;
        ri++;
       }
       else
        while (!feof(fp) && (fgetc(fp) != '\n'));
       }
     fclose(fp);
     if (ni < ri)
       error = TRUE;
     if ((Ti == 0) || (ri == 0))
      error = TRUE;
   }
   else
   {
     printf("Check_Data_File: can't open data file  %s\n", c);
     error = TRUE;
   }
   if (!error)
   {
     X1[Samples_NO] = minimum_t;
     Yi[Samples_NO]=(Ti-ri*X1[Samples_NO])+(ni-ri)*(maximum_t-X1[Samples_NO]);
     Sample_Size[Samples_NO] = ni;
     Sample_ri[Samples_NO] = ri;
     Samples_NO++;
   }
 }
 return (!error);
}
```

```
/*******************************************************
** DATE: 31/01/1992                                   **
** VERSION: 1.0                                       **
** MODULE NAME: Compute_LRT()                         **
** MODULE NUMBER: 1.2.3                               **
** DESCRIPTION: This module is used to compute the test*
**      criteria from the global variables            **
**      (Samples_NO,Sample_Size, and Sample_ri).      **
    **
** PASSED VARIABLES:                                  **
** RETURNS: The value of LRT                          **
** GLOBAL VARIABLES USED: Sample_Size[], Sample_ri[], **
**        Samples_NO,total # of failures R, X1[], Yi[]**
   **
**        and LRT                                     **
** GLOBAL VARIABLES CHANGED: R, LRT                   **
** FILES READ: None                                   **
** FILES WRITTEN: None                                **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT:None                               **
** MODULES CALLED: None                               **
** CALLING MODULES: Compare_Sets()                    **
** AUTHOR: Salah Amin Elewa                           **
** HISTORY:                                           **
**         1.0 Salah Amin Elewa 01/31/1992            **
**             original  version                      **
*******************************************************/
double Compute_LRT(void)
{
  double   part1, part2, Yitot, V, u;
  int      i;
  double   Zp = 1.0e37;
  part1 = part2 = LRT = 0.0;
```

```
  V = u = Yitot = 0.0;
  if (Samples_NO >= 2)
  {
    for (i = 0; i < Samples_NO; i++)
      if (X1[i] < Zp)
        Zp = X1[i];
    for (i = 0; i < Samples_NO; i++)
    {
      R = R + Sample_ri[i];
      u = u + Sample_Size[i] * (X1[i] - Zp);
      Yitot = Yitot + Yi[i];
    }
    V = Yitot + u;
    part1 = (double) R *log(((double) R / V));
    part2 = 0.0;
    for (i = 0; i < Samples_NO; i++)
     part2=part2+(double)Sample_ri[i]*log(Yi[i]/(double)Sample_ri[i]);
    LRT = exp(part1 + part2);
  }
  return (LRT);
}
```

```
/*******************************************************
** DATE: 01/31/1992                                   **
** VERSION: 1.0                                       **
** MODULE NAME: Print_Results()                       **
** MODULE NUMBER: 1.2.4                               **
** DESCRIPTION: This module is used to print the resu-**
**      lting table on both the screen and the output **
**      file "report.dat".                            **
** PASSED VARIABLES: L, Failure Data Set number       **
** RETURNS: None                                      **
** GLOBAL VARIABLES USED: DIFFERENT                   **
** GLOBAL VARIABLES CHANGED: DIFFERENT                **
** FILES READ: None                                   **
** FILES WRITTEN: "report.dat" pointed to by fp5      **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT: Screen                            **
** MODULES CALLED: Compute_Percentile                 **
** CALLING MODULES: Compare_Sets()                    **
** AUTHOR: Salah Amin Elewa                           **
** HISTORY:                                           **
**          1.0 Salah Amin Elewa     01/31/1992       **
**              original  version                     **
*******************************************************/
void Print_Results(double L, int J)
{
  int     i, j, k;
  double  alpha[MAX_ALPHA]={.00025,.0005,.001,.0025,.005,.01,.025,.05};
  double  C[MAX_ALPHA], prb;
  char    *SET_NAME[DATA_FILES]={" SET_1 "," SET_2 "," SET_4 "," SET_5 ",
  " SET_6 ","SET_14c","SET_17 ","SET_27 ","SET_S1a","SET_S1c",
  "SET_SS3","SET_SS4","SET_Lt2","SET_Lt3"};

  if (L > .999)
  {
```

```
   DIFFERENT = FALSE;
   printf("\n ├──────┼");
   fprintf(fp5,"\n ├──────┼");
   for (j = 1; j < 8; j++)
     printf(" ──────┴");
   printf("──────┤\n");

   for (j = 1; j < 8; j++)
     fprintf(fp5, "──────┴");
   fprintf(fp5, "──────┤\n");

   printf("|%s|", SET_NAME[J - 1]);
   fprintf(fp5, "|%s|", SET_NAME[J - 1]);

   printf("     Two Identical Sets or Set Compared with Itself L = 1.0     |");
   fprintf(fp5,"     Two Identical Sets or Set Compared with Itself L = 1.0
|");

   if (J < (DATA_FILES - 2))
   {
     printf("\n ├──────┼");
     fprintf(fp5, "\n ├──────┼");

     for (j = 1; j < 8; j++)
      printf("──────┬");
     printf("──────┤\n");

     for (j = 1; j < 8; j++)
       fprintf(fp5, "──────┬");
     fprintf(fp5, "──────┤\n");
   }
 }
 else
 {
```

```
if (DIFFERENT)
{
  printf("\n├");
  fprintf(fp5, "\n├");

  for (j = 1; j < 9; j++)
  printf("───────┼");
  printf("───────┤\n");

  for (j = 1; j < 9; j++)
 fprintf(fp5, "───────┼");
  fprintf(fp5, "───────┤\n");
}

printf("|%s|", SET_NAME[J - 1]);
fprintf(fp5, "|%s|", SET_NAME[J - 1]);
for (i = 0; i < MAX_ALPHA; i++)
{
  Compute_Percentile(alpha[i],L,&C[i],&prb);
  if (L < C[i])
  {
   printf("N/%3.2lf|", L - C[i]);
    fprintf(fp5, "N/%3.2lf|", L - C[i]);
   for (k = i; k < 7; k++)
   {
     printf("       |");
     fprintf(fp5, "       |");
   }
     i = 8; /* not to recalculate if it is rejected */
  }
  else
  {
   printf("Y/%4.3lf|", L - C[i]);
   fprintf(fp5, "Y/%4.3lf|", L - C[i]);
```

```
    }
  }
  printf("\n|       |");
  fprintf(fp5, "\n|       |");
  for (i = 0; i < MAX_ALPHA; i++)
  {
    if (L < C[i])
    {
      printf("L=%4.3lf|", L);
      fprintf(fp5, "L=%4.3lf|", L);
      for (k = i; k < 7; k++)
      {
        printf("       |");
        fprintf(fp5, "       |");
      }
      i = 8; /* not to recalculate if it is rejected */
    }
    else
    {
      printf("L=%4.3lf|", L);
      fprintf(fp5, "L=%4.3lf|", L);
    }
  }
  printf("\n|       |");
  fprintf(fp5, "\n|       |");
  for (i = 0; i < MAX_ALPHA; i++)
  {
    if (L < C[i])
    {
      printf("C=%4.3lf|", C[i]);
      fprintf(fp5, "C=%4.3lf|", C[i]);
      for (k = i; k < 7; k++)
      {
        printf("       |");
```

```
        fprintf(fp5, "        |");
       }
        i = 8; /* not to recalculate if it is rejected */
      }
      else
      {
        printf("C=%4.3lf|", C[i]);
       fprintf(fp5, "C=%4.3lf|", C[i]);
      }
    }
    DIFFERENT = TRUE;
  }
  return;
}
```

```
/***********************************************************
** DATE: 01/31/1992                                      **
** VERSION: 2.0                                          **
** MODULE NAME: Compute_Percentile()                     **
** MODULE NUMBER: 1.2.5                                  **
** DESCRIPTION: This module is used to compute the       **
**       percentage point necessary for evaluating the   **
**       LRT criteria at the values given by the global**
**       variables Sample_ri, total failures, and the   **
**       significance level alpha                       **
** PASSED VARIABLES: alpha,LRT                           **
** RETURNS:values stored in the locations pointed to by*
**    pct - percentage point for alpha value given, and**
**    prb - probability value.                          **
** GLOBAL VARIABLES USED: ka[],R,LRT,Samples_NO,        **
**                        Sample_ri                     **
** GLOBAL VARIABLES CHANGED: ka[]                        **
** FILES READ: None                                      **
** FILES WRITTEN: None                                   **
** MODULES CALLED: Prob(),Newton(),Ri_Coef(),Ar_Coef()**
** CALLING MODULES: Print_Results()                      **
** AUTHOR: Salah Amin Elewa                              **
** HISTORY:1.0 Capt. K. N. Cole 5/8 /1985                **
**              original  version                        **
**          1.1 Capt. K. N. Cole 10/25 /1986             **
**              modified for SAE system                  **
**          2.0 Salah Amin Elewa      01/31/1992         **
**              - header modified                        **
**              - names for module and some variables   **
**                changed                                **
**              - program modified to work with 2-para-**
**                meter exponential distribution and to**
**                read sample size from file "input.dat"*
***********************************************************/
```

```
void
Compute_Percentile(double alpha,double LRT,double *pct,double *prb)
{
  extern double  Ar[DATA_FILES + 1];
  extern double  Prob(double,double,double,int,double,double);
  extern double  Newton(double,double,double,double,int,double,double);
  extern         Ar_Coef(double, double);
  extern void    Ri_Coef(double, double, double, double);
  double         v, a, t, p, sum, m, delta;
  int            i;

  m = delta = sum = a = 0.0;
  p = (double) Samples_NO;
  for (i = 0; i < Samples_NO; i++)
    ka[i] = (double) Sample_ri[i] / (double) R;
  v = 3.0 * (p - 1.0) / 2.0;
  t = (p + 1.0) / (6.0 * p);
  for (i = 0, sum = 0.0; i < Samples_NO; i++)
    sum = sum + 1.0 / ka[i];
  delta = (13.0 * (sum - 1.0)) / ((p - 1.0) * 18.0);
  a = (1.0 - v) /2.0;
  m = ((double) R - delta) / p;
  Ri_Coef(p, delta, a, v); /*initialize coefficients*/
  *prb = Prob(a, v, p, R, LRT, m);
  *pct = Newton(a, v, t, p, R, alpha, m);
  return;
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.0                                       **
** TITLE: Probability Computation  File               **
** FILENAME: Prob.c                                   **
** COORDINATOR: Salah Amin Elewa                      **
** PROJECT:Development of an Environment for Software **
**                 Reliability Model Selection.       **
** OPERATING SYSTEM:MS DOS version 2.0 or higher      **
** LANGUAGE: Turbo C (2.1)                            **
** FILE PROCESSING: Compile and link with files prep.c**
**     coefs.c, compute.c, and pms.c                  **
** CONTENTS:                                          **
**          1.3.1 - Prob()                            **
**          1.3.2 - Newton()                          **
**          1.3.3 - lnGamma()                         **
**          1.4.4 - Ix()                              **
**          1.4.5 - Betacf()                          **
**          1.4.6 - Inv_Beta()                        **
** FUNCTION:This file contains the modules for Comput-**
**          ing the distribution function and calcula-**
**          ting  the percentage point using Newton's **
**          approximation method                      **
*******************************************************/
#include <stdio.h>
#include <math.h>
#include "defs.h"

boolean  exact=TRUE;
extern double   Ri[], ka[];
double Prob(double,double,double,int,double,double),lnGamma(double);
double Newton(double,double,double,double,int,double,double);

static double   Ix(double, double, double);
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.0                                       **
** MODULE NAME: Prob()                                **
** MODULE NUMBER: 1.4.1                               **
** DESCRIPTION: This module is used for computing the **
**       distribution function of x                   **
** PASSED VARIABLES: a, v, p, R, x, m                 **
** RETURNS:Prob (double)                              **
** GLOBAL VARIABLES USED: Ri[], ka[], exact, R        **
** GLOBAL VARIABLES CHANGED: None                     **
** FILES READ: None                                   **
** FILES WRITTEN:None                                 **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT:None                               **
** MODULES CALLED: lnGamma(), Ix()                    **
** CALLING MODULES: Compute_Percentile(), Newton()    **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**       1.0 Salah A. Elewa 01/16/1992                **
**           original version                         **
*******************************************************/
double
Prob(double a,double v,double p,int R,double x,double m)
{
  int    i , k;
  double Rdsh,tsum,t2, K3;
  tsum=Rdsh=0.0;
  if (exact)
  {
    for (k = 0, t2 = 0.0; k < (int) p; k++)
     t2=t2+((((double)R*ka[k]-1.5)*log(ka[k]))-lnGamma((double)R*ka[k]-1.0));
    K3=exp(lnGamma((double)R-1.0)+t2-v*log(p)+0.5*(p-1.0)*log(2.0*3.14159));
    for (i = 0, t2 = 0.0; i <= limit; i++)
```

```
    {
      Rdsh=exp(lnGamma((double)m+a)-lnGamma((double)m+a+v+i));
      t2 = t2 + Ri[i] * Rdsh * Ix((double) m + a, v + i, x);
    }
    tsum = K3 * t2;
  }
  else
    tsum=Ix(m+a,v,x)+Ri[2]*(Ix(m+a,v+2,x)-Ix(m+a,v,x))/m/m;
  return (tsum);
}
```

```
/*****************************************************
** DATE: 01/16/1992                                  **
** VERSION: 1.1                                      **
** MODULE NAME: Newton()                             **
** MODULE NUMBER: 1.4.2                              **
** DESCRIPTION:This module is used for computing the **
**            percentage point using Newton's approx-**
**            imation method                         **
** PASSED VARIABLES: a, v, p, t, R, alpha, m         **
** RETURNS:Newton (double)                           **
** GLOBAL VARIABLES USED: DATA_FILES, R              **
** GLOBAL VARIABLES CHANGED: None                    **
** FILES READ: None                                  **
** FILES WRITTEN:None                                **
** HARDWARE INPUT: None                              **
** HARDWARE OUTPUT:None                              **
** MODULES CALLED: Prob(), Inv_Beta()                **
** CALLING MODULES: Compute_Percentile(), Process_Equal*
**                  and Process_Unequal in table.c   **
** AUTHOR: Salah A. Elewa                            **
** HISTORY:                                          **
**       1.0 Capt. K. N. Cole 01/05/1985             **
**           Initial Version                         **
**       1.1 Salah A. Elewa   01/16/1992             **
**           Header modified, Variable names changed **
*****************************************************/
double
Newton(double a,double v,double t,double p,int R,double alpha,double m)

{
  double   Inv_Beta(double, double, double);
  double   z[NEWTON_LIMIT + 2], pa[NEWTON_LIMIT + 2];
  double   sp, x;
  int      k, done;
```

```
    z[1] = Inv_Beta(alpha, (double) R - t, v);
    if (p <= 3.0)
      z[2] = z[1] + 0.05;
    else if (z[1] > 0.05)
      z[2] = z[1] - 0.05;
    else
      z[2] = z[1] / 2.0;
    x = z[1];
    sp = Prob(a, v, p, R, x,m);
    pa[1] = sp;
    done = FALSE;
    for (k = 2; (k <= NEWTON_LIMIT) && !done; k++)
    {
      if (z[k] > 1.0)
        z[k] = z[k - 1] + ((1 - z[k - 1]) / 2.0);
      x = z[k];
      sp = Prob(a, v, p, R, x,m);
      pa[k] = sp;
      if (fabs(sp - alpha) < 0.0000001)
        done = TRUE;
      else
        z[k+1] = z[k]-(z[k]-z[k-1])*(sp-alpha)/(sp-pa[k-1]);
      if (z[k + 1] < 0.0)
        z[k + 1] = z[k] / 2.0;
      if ((sp > 1.0) || (sp < 0.0))
      {
        printf("\nNewton.c:Incorrect Prob Value %f %f \n",x,sp);
        done = TRUE;
      }
    }
    return (z[k - 1]);
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.1                                       **
** MODULE NAME: lnGamma()                             **
** MODULE NUMBER: 1.4.3                               **
** DESCRIPTION:This module is used for computing the  **
**              the natural log of the Gamma function **
** PASSED VARIABLES: dx                               **
** RETURNS:lnGamma (double)                           **
** GLOBAL VARIABLES USED: None                        **
** GLOBAL VARIABLES CHANGED: None                     **
** FILES READ: None                                   **
** FILES WRITTEN:None                                 **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT:None                               **
** MODULES CALLED: None                               **
** CALLING MODULES: Prob(), Ix()                      **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**       1.0 Capt. K. N. Cole 01/05/1985              **
**           Translated from Pascal code              **
**           (Original by Capt. Mark Amell)           **
**       1.1 Salah A. Elewa   01/16/1992              **
**           Header modified, module name changed     **
*******************************************************/
double lnGamma(double dx)
{
  double    rdo, dy, dterm, de, da, db, domeg, dlggm;
  double    ds, dz, dw, dv, du, dt, dr, dq, dp;

  rdo = 0.0;
  dy = dx;
  dterm = 1.0;
  de = 1.0;
```

```
  domeg = 1.0e25;
  da = 0.9999999999;
  db = 1.0000000001;
  dlggm = domeg;
  if (dx >= rdo)
  {
    dlggm = rdo;
    if ((dx <= da) || (dx >= db))
    {
      if ((dx <= (da + de)) || (dx >= (db + de)))
      {
       while ((dy - 18.00) <= 0.0)
       {
         dterm = dterm * dy;
         dy = dy + de;
       }
       ds = de / (dy * dy);
       dz = (double) 0.005410256410256410;
       dw = (double) -0.001917526917526918;
       dv = (double) 0.0008417508417518418;
       du = (double) -0.0005952380952360952;
       dt = (double) 0.0007936507936507937;
       dr = (double) -0.002777777777777778;
       dq = (double) 0.08333333333333333;
       dp = (double) 0.9189385332046727;
       dlggm = ((dy-0.5)*log(dy))+dp-dy-log(dterm);
       dlggm = dlggm+((((((dz*ds+dw)*ds+dv)*ds+du)*
       ds + dt) * ds + dr) * ds + dq) / dy;
      }
    }
  }
  return (dlggm);
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.1                                       **
** MODULE NAME: Ix()                                  **
** MODULE NUMBER: 1.4.4                               **
** DESCRIPTION:This module is used for calculating the**
**             Incomplete Beta function using the con-**
**             tinued fraction method.                **
** PASSED VARIABLES: a,b,x                            **
** RETURNS:Ix(double)                                 **
** GLOBAL VARIABLES USED: None                        **
** GLOBAL VARIABLES CHANGED: None                     **
** FILES READ: None                                   **
** FILES WRITTEN:None                                 **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT:None                               **
** MODULES CALLED: Betacf(), lnGamma()                **
** CALLING MODULES: Prob(), Inv_Beta()                **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**       1.0 Taken from "Numerical Recipes in C" by   **
**           William H. Press and Others, Cambridge   **
**           University Press,pp.179,1990.            **
**       1.1 Salah A. Elewa   01/16/1992              **
**           Header Added, nrerror() function replaced**
**           by printf() function                     **
*******************************************************/
double Ix(double a,double b,double x)
{
  double  Betacf(double a, double b, double x);
  double  bt;

  if (x < 0.0 || x > 1.0)
    printf("Bad x in routine Ix()\n");
```

```
  if (x == 0.0 || x == 1.0)
    bt = 0.0;
  else   /* Factors in front of the continued fraction */
    bt=exp(lnGamma(a+b)-lnGamma(a)-lnGamma(b)+a*log(x)+b*log(1-x));
  if(x<(a+1.0)/(a+b+2.0))/* Use continued fractions directly */
    return bt * Betacf(a, b, x) / a;
  else
    return 1.0 - bt * Betacf(b, a, 1.0 - x) / b;
  /*Use continued fractions after making symmetry transformation*/
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.1                                       **
** MODULE NAME: Betacf()                              **
** MODULE NUMBER: 1.4.5                               **
** DESCRIPTION:This module is used for evaluating the **
**             continued fractions for the Ix() funct-**
**             tion above.                            **
** PASSED VARIABLES: a,b,x                            **
** RETURNS:Betacf(double)                             **
** GLOBAL VARIABLES USED: None                        **
** GLOBAL VARIABLES CHANGED: None                     **
** FILES READ: None                                   **
** FILES WRITTEN:None                                 **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT:None                               **
** MODULES CALLED: None                               **
** CALLING MODULES: Ix()                              **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**       1.0 Taken from "Numerical Recipes in C" by   **
**           William H. Press and Others, Cambridge   **
**           University Press,pp.180,1990.            **
**       1.1 Salah A. Elewa   01/16/1992              **
**           Header Added, nrerror() function replaced**
**           by printf() function                     **
*******************************************************/
double Betacf(double a, double b, double x)
{
  double   qap, qam, qab, em, tem, d;
  double   bz, bm = 1.0, bp, bpp;
  double   az = 1.0, am = 1.0, ap, app, aold;
  int      m;
```

```
  qab = a + b;
  qap = a + 1.0;
  qam = a - 1.0;
  bz = 1.0 - qab * x / qap;
  for (m = 1; m <= ITMAX; m++)
  {
    /* continued fraction evaluation by the recurrence method */
    em = (double) m;
    tem = em + em;
    d = em * (b - em) * x / ((qam + tem) * (a + tem));
    ap = az + d * am;
    /* One step (the even one) of the recurrence */
    bp = bz + d * bm;
    d = -(a + em) * (qab + em) * x / ((qap + tem) * (a + tem));
    app = ap + d * az;
    /* Next step of the recurrence (the odd one) */
    bpp = bp + d * bz;
    aold = az;            /* Save the old answer */
    am = ap / bpp;    /* Renormalize to prevent overflows */
    bm = bp / bpp;
    az = app / bpp;
    bz = 1.0;
    if (fabs(az - aold) < (EPS * fabs(az)))
      return az;                 /* Are we done */
  }
  printf("a or b too big or ITMAX too small in BETACAF \n");
  return 0.0;
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.1                                       **
** MODULE NAME: lnv_Beta                              **
** MODULE NUMBER: 1.4.6                               **
** DESCRIPTION:This module is used for computing the  **
**             inverse Beta function                  **
** PASSED VARIABLES: alpha, p, q                      **
** RETURNS:Inv_Beta(double)                           **
** GLOBAL VARIABLES USED: None                        **
** GLOBAL VARIABLES CHANGED: None                     **
** FILES READ: None                                   **
** FILES WRITTEN:None                                 **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT:None                               **
** MODULES CALLED: Ix()                               **
** CALLING MODULES: Newton()                          **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**       1.0 Capt. K. N. Cole 01/05/1985              **
**           Translated from Pascal code              **
**           (Original by Capt. Mark Amell)           **
**       1.1 Salah A. Elewa   01/16/1992              **
**           Header modified, module name changed     **
*******************************************************/
double Inv_Beta(double alpha, double p, double q)
{
  int      jj,jend,j,i;
  double   dp,dl,dif,dlx,dux,dmp,decr,dmpu,dm,dn,du,dfd,
   dabf,dfune,esp1,esp2,esp3,esp4;
  boolean  flag;
  double   darg[5],dfun[5];

  esp1 = 1.0e-180;
```

```
esp2 = 1.0e-13;
esp3 = 1.0e-11;
esp4 = 1.0e-10;
dp = alpha;
dm = p;
flag = TRUE;
dn = q;
du = 1.0;
if (((dp * (du-dp))<0.0)||((dm<0.0)||(dn<0.0)))
  dmp = 0.0;
else if ((dp * (du - dp)) == 0.0)
  dmp = alpha;
else if (dm == 1.0)
  dmp = du - exp((du / dn) * log(du - dp));
else if (dn == 1.0)
  dmp = exp((du / dm) * log(dp));
else
{
  flag = FALSE;
  dl = 0.0;
  dif = 1.0 / 3.0;
  dlx = -dp;
  dux = du - dp;
  jj = 0;
  dmpu = 0.0;
  jend = 3;
  while ((jj < 25) && (!flag))
  {
    if (jj == 25)
    jend = 3;
    jj = jj + 1;
    j = 1;
    while ((j <= jend) && (!flag))
    {
```

```
dmp = (du + dl) / 2.0;
i = 1;
if (((du-dl)<esp1)||(((du-dl)<(esp2*dp))&&(dl>esp2)))
flag = TRUE;
 else
while ((i < 3) && (!flag))
{
  darg[i] = dl + (du - dl) * dif * i;
  dfun[i] = Ix(dm, dn, darg[i]) - dp;
  if (dfun[i] == 0)
    dmp = darg[i];
  if (dfun[i] == 0)
    flag = TRUE;
  else if ((dfun[i] < 0.0) && (i == 2))
  {
    dl = darg[2];
    dlx = dfun[2];
  }
  else if (dfun[i] > 0.0)
  {
    du = darg[i];
    dux = dfun[i];
    if (i == 2)
    {
      dl = darg[1];
      dlx = dfun[1];
    }
     else
       i = 2;
  }
  ++i;
 }
 ++j;
}
```

```
if (!flag)
{
 jend = 2;
dmp = (du + dl) / 2.0;
dfd = dux - dlx;
if ((dfd < esp3) && (dfd < (esp4 * dp)))
flag = TRUE;
}
if (!flag)
{
decr = dux * (du - dl) / dfd;
dmp = du - decr;
if(((dmp-dl)<esp1)||(((dmp-dl)<esp2)&&(dl>esp2)))
flag = TRUE;
if (!flag)
 {
   dfun[3] = Ix(dm, dn, dmp) - dp;
   dabf = fabs(dfun[3]);
   dfune = dfun[3];
   if(((dabf<esp3)&&(dabf<(esp4*dp)))||((dmp<esp1)||
   (((du-dmpu)<esp2)&&(du>0.999999999999))||
   (dfun[3] == 0.0)))
   flag = TRUE;
  }
 if (!flag)
 {
   if (dfun[3] < 0.0)
   {
     if (decr < (0.9 * (du - dl)))
     {
       dl = dmp;
       dlx = dfune;
     }
     else
```

```
        {
          dmpu = dmp;
          dmp = 5.0 * (dmp - dl) + dl;
          dfune = Ix(dm, dn, dmp) - dp;
          if (dfune == 0.0)
            flag = TRUE;
          if (!flag)
          {
           if (dfune < 0.0)
           {
             dl = dmp;
             dlx = dfune;
           }
           else
           {
             du = dmp;
             dux = dfune;
             dl = dmpu;
             dlx = dfun[3];
           }
          }
        }
       }
      else
      {
        if (decr >= (0.1 * (du - dl)))
        {
          du = dmp;
          dux = dfune;
        }
        else
        {
          dmpu = dmp;
          dmp = du - 5.0 * decr;
```

```
          dfune = Ix(dm, dn, dmp) - dp;
          if (dfune == 0.0)
          flag = TRUE;
          if (!flag)
          {
            if (dfune < 0.0)
            {
              du = dmpu;
              dux = dfun[3];
              dl = dmp;
              dlx = dfune;
            }
            else
            {
              du = dmp;
              dux = dfune;
            }
          }
        }
      }
    }
   }
  }
 }
 return (dmp);
}
```

```
/*****************************************************
** DATE:01/16/1992                                  **
** VERSION: 1.0                                     **
** TITLE: Coefficients file                         **
** FILENAME: coefs.c                                **
** COORDINATOR: Salah Amin Elewa                    **
** PROJECT:Development of an Environment for Software **
**                 Reliability Model Selection.     **
** OPERATING SYSTEM: MS DOS version 2.3 or higher   **
** LANGUAGE: Turbo C  (2.1)                         **
** FILE PROCESSING: Compile and link with files pms.c **
**     compute.c, prep.c, and  prob.c.              **
** CONTENTS:                                        **
**           1.4.1 - Ri_Coef()                      **
**           1.4.2 - q_Coef()                       **
**           1.4.3 - Ar_Coef()                      **
**           1.4.4 - Create_Dir()                   **
**           1.4.5 - Create_Cir()                   **
**           1.4.6 - Bern_Poly()                    **
**           1.4.7 - Bern_Num()                     **
**           1.4.8 - com()                          **
** FUNCTION: This file includes the routines used for **
**      creating the different coefficients needed by **
**      the environment.                            **
*****************************************************/
#include <math.h>
#include "defs.h"

boolean  bflag = FALSE;
double   ka[MAX_SAMPLES + 1];
double   Ri[DATA_FILES + 1], Ar[DATA_FILES + 1];
void     Ar_Coef(double, double);
void     Ri_Coef(double, double, double, double);
```

```
static double   com(int, int);
static double   Bern_Poly(int, double);
static double   q[DATA_FILES+1],barray[DATA_FILES+2];
static double   Cir[DATA_FILES + 1][DATA_FILES + 1];
static double   Dir[DATA_FILES + 1][DATA_FILES + 1];
```

```
/*****************************************************
** DATE: 01/16/1992                                 **
** VERSION: 1.1                                     **
** MODULE NAME: Ri_Coef()                           **
** MODULE NUMBER: 1.4.1                             **
** DESCRIPTION: This module is used for creating the **
**       Ri coefficients needed by Process_Equal and **
**       Process_Unequal in table.c and Prob() module **
** PASSED VARIABLES: p,delta,a,v                    **
** RETURNS:None                                     **
** GLOBAL VARIABLES USED: q[], Dir[], Ri[], DATA_FILES**
** GLOBAL VARIABLES CHANGED: Ri[]                   **
** FILES READ: None                                 **
** FILES WRITTEN: None                              **
** HARDWARE INPUT: None                             **
** HARDWARE OUTPUT: None                            **
** MODULES CALLED: Create_Dir(), q_Coef()           **
** CALLING MODULES: Process_Equal and Process_Unequal **
**           in table.c and Compute_Percentile()    **
** AUTHOR: Salah A. Elewa                           **
** HISTORY:                                         **
**       1.0 Capt. K. N. Cole 08/05/1985            **
**           Translated from Pascal code            **
**           (Original by Capt. Mark Amell)         **
**       1.1 Salah A. Elewa   01/16/1992            **
**           Header modified, module names changed  **
*****************************************************/
void Ri_Coef(double p, double delta, double a, double v)
{
  void    q_Coef(double, double);
  void    Create_Dir(double, double);
  double  sum;
  int     i, k;
```

```
  Create_Dir(a, v);
  q_Coef(p, delta);
  Ri[0] = 1.0;
  for (i = 1; i <= DATA_FILES; i++)
  {
    for (sum = 0.0, k = 1; k <= i; k++)
      sum = sum + Ri[i - k] * Dir[i - k][k];
    Ri[i] = (q[i] - sum) / Dir[i][0];
  }
  return;
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.1                                       **
** MODULE NAME: q_Coef()                              **
** MODULE NUMBER: 1.4.2                               **
** DESCRIPTION:This module is used for calculating the**
**              q coefficients for module Ri_Coef     **
** PASSED VARIABLES: p,delta                          **
** RETURNS:None                                       **
** GLOBAL VARIABLES USED: Ar[], q[], DATA_FILES       **
** GLOBAL VARIABLES CHANGED: q[]                      **
** FILES READ: None                                   **
** FILES WRITTEN: None                                **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT: None                              **
** MODULES CALLED: Ar_Coef()                          **
** CALLING MODULES: Ri_Coef()                         **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**        1.0 Capt. K. N. Cole 08/05/1985             **
**           Translated from Pascal code              **
**           (Original by Capt. Mark Amell)           **
**       1.1 Salah A. Elewa   01/16/1992              **
**           Header modified, module names changed    **
*******************************************************/
void q_Coef(double p, double delta)
{
  int     i, k;
  double  sum;

  Ar_Coef(p, delta);
  q[0] = 1.0;
  for (i = 1; i <= DATA_FILES; i++)
  {
```

```
    for (sum = 0.0, k = 1; k <= i; k++)
      sum = sum + k * Ar[k] * q[i - k];
    q[i] = (1.0 / (double) i) * sum;
  }
  return;
}
```

```
/*****************************************************
** DATE: 01/16/1992                                 **
** VERSION: 1.2                                     **
** MODULE NAME: Ar_Coef()                           **
** MODULE NUMBER: 1.4.3                             **
** DESCRIPTION:This module is used for calculating the**
**             Ar coefficients for module q_Coef    **
** PASSED VARIABLES: p,delta                        **
** RETURNS:None                                     **
** GLOBAL VARIABLES USED: Ar[], ka[], DATA_FILES    **
** GLOBAL VARIABLES CHANGED: Ar[]                   **
** FILES READ: None                                 **
** FILES WRITTEN: None                              **
** HARDWARE INPUT: None                             **
** HARDWARE OUTPUT: None                            **
** MODULES CALLED: Bern_Poly()                      **
** CALLING MODULES: q_Coef and Process_Equal and    **
**         Process_Unequal in table.c               **
** AUTHOR: Salah A. Elewa                           **
** HISTORY:                                         **
**        1.0 Capt. K. N. Cole 01/12/1984           **
**                (Initial Version)                 **
**        1.1 Capt. K. N. Cole 05/10/1985           **
**            Modified for new formula              **
**        1.2 Salah A. Elewa   01/16/1992           **
**           Header modified, module names changed  **
*****************************************************/
void Ar_Coef(double p, double delta)
{
 double  factor, partial;
 int     i, k;

 Ar[0]  = 1.0;
 factor = 1.0;
```

```
for (k = 1; k <= DATA_FILES; k++)
{
  factor = factor * -1.0 / p;
  for (partial = 0.0, i = 0; i < p; i++)
    partial=partial+Bern_Poly(k+1,(delta*ka[i]-1))/exp(log(ka[i])*k);
  Ar[k] =(factor/(double)(k*(k+1)))*(partial-Bern_Poly(k+1,(delta-1)));
}
return;
}
```

```
/*******************************************************
** DATE: 01/16/1992                                  **
** VERSION: 1.2                                      **
** MODULE NAME: Create_Dir()                         **
** MODULE NUMBER: 1.4.4                              **
** DESCRIPTION:This module is used for creating the  **
**             Dir coefficients for module Ri_Coef   **
** PASSED VARIABLES: a,v                             **
** RETURNS:None                                      **
** GLOBAL VARIABLES USED: Cir[], Dir[], DATA_FILES   **
** GLOBAL VARIABLES CHANGED: Dir[]                   **
** FILES READ: None                                  **
** FILES WRITTEN: None                               **
** HARDWARE INPUT: None                              **
** HARDWARE OUTPUT: None                             **
** MODULES CALLED: Create_Cir()                      **
** CALLING MODULES: Ri_Coef()                        **
** AUTHOR: Salah A. Elewa                            **
** HISTORY:                                          **
**       1.0 Capt. Mark F. Amell 01/12/1984          **
**               (Initial Version)                   **
**       1.1 Capt. K. N. Cole    05/10/1985          **
**           Header added                            **
**       1.2 Salah A. Elewa   01/16/1992             **
**          Header modified, module names changed    **
*******************************************************/
void Create_Dir(double a, double v)
{
  void   Create_Cir(double, double);
  int    r, i, k;
  double sum;

  Create_Cir(a, v);
  for (i = 0; i <= DATA_FILES; i++)
```

```
    Dir[i][0] = 1.0;
  for (r = 1; r <= DATA_FILES; r++)
  {
    for (i = 0; i <= DATA_FILES; i++)
    {
      for (sum = 0.0, k = 1; k <= r; k++)
      sum = sum + k * Cir[i][k] * Dir[i][r - k];
      Dir[i][r] = sum / (double) r;
    }
  }
  return;
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.2                                       **
** MODULE NAME: Create_Cir()                          **
** MODULE NUMBER: 1.4.5                               **
** DESCRIPTION:This module is used for creating the   **
**           Cir coefficients for module Create_Dir()**
** PASSED VARIABLES: a,v                              **
** RETURNS:None                                       **
** GLOBAL VARIABLES USED: Cir[], DATA_FILES           **
** GLOBAL VARIABLES CHANGED: Cir[]                    **
** FILES READ: None                                   **
** FILES WRITTEN: None                                **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT: None                              **
** MODULES CALLED: Bern_Poly()                        **
** CALLING MODULES: Create_Dir()                      **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**       1.0 Capt. Mark F. Amell 01/12/1984           **
**               (Initial Version)                    **
**       1.1 Capt. K. N. Cole    05/10/1985           **
**           Header added                             **
**       1.2 Salah A. Elewa   01/16/1992              **
**          Header modified, module names changed     **
*******************************************************/
void Create_Cir(double a, double v)
{
  int     r, i, rt;
  double  sign, temp;

  for (i = 0; i <= DATA_FILES; ++i)
    Cir[i][0] = 1.0;
```

```
  for (sign = 1.0, r = 1; r <= DATA_FILES; ++r)
  {
    rt = r + 1;
    temp = Bern_Poly(rt, a);
    for (i = 0; i <= DATA_FILES; ++i)
      Cir[i][r]=(sign/(r*(r+1.0)))*(temp-Bern_Poly(rt,a+v+i));
    sign = sign * (-1.0);
  }
  return;
}
```

```
/*********************************************************
** DATE: 01/16/1992                                     **
** VERSION: 1.1                                         **
** MODULE NAME: Bern_Poly()                             **
** MODULE NUMBER: 1.4.6                                 **
** DESCRIPTION:This module is used for creating the     **
**           Bernoulli polynomials                      **
** PASSED VARIABLES: n,x                                **
** RETURNS:Bern_Poly (double)                           **
** GLOBAL VARIABLES USED: barray[]                      **
** GLOBAL VARIABLES CHANGED: None                       **
** FILES READ: None                                     **
** FILES WRITTEN: None                                  **
** HARDWARE INPUT: None                                 **
** HARDWARE OUTPUT: None                                **
** MODULES CALLED: Bern_Num(), com()                    **
** CALLING MODULES: Ar_Coef(), Create_Cir()             **
** AUTHOR: Salah A. Elewa                               **
** HISTORY:                                             **
**      1.0 Capt. K. N. Cole 08/05/1985                 **
**          Translated from Pascal code                 **
**          (Original by Capt. Mark Amell)              **
**      1.1 Salah A. Elewa   01/16/1992                 **
**          Header modified, module name changed        **
*********************************************************/
double  Bern_Poly(int n, double x)
{
  void    Bern_Num(void);
  int     i;
  double  sum, power;

  sum = 0.0;
  power = 1.0;
```

```
  if (!bflag)
  {
    bflag = TRUE;
    Bern_Num();  /* initialize barray */
  }
  for (i = n; i >= 0; i--)
  {
    sum = sum + com(n, i) * barray[i] * power;
    power = power * x;
  }
  return (sum);
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.1                                       **
** MODULE NAME: Bern_Num()                            **
** MODULE NUMBER: 1.4.7                               **
** DESCRIPTION:This module calculates the Bernoulli   **
**            numbers and stores them in array (barray)**
** PASSED VARIABLES: None                             **
** RETURNS:None                                       **
** GLOBAL VARIABLES USED: barray[], DATA_FILES        **
** GLOBAL VARIABLES CHANGED: barray[]                 **
** FILES READ: None                                   **
** FILES WRITTEN: None                                **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT: None                              **
** MODULES CALLED: com()                              **
** CALLING MODULES: Bern_Poly()                       **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**        1.0 Capt. K. N. Cole 08/05/1985             **
**            Translated from Pascal code             **
**            (Original by Capt. Mark Amell)          **
**        1.1 Salah A. Elewa   01/16/1992             **
**            Header modified, module name changed    **
*******************************************************/
void Bern_Num(void)
{
  double  sum;
  int     i, j;

  barray[0] = 1.0;
  for (j = 1; j <= (DATA_FILES + 1); j++)
  {
    sum = 0.0;
```

```
    for (i = 0; i <= (j - 1); i++)
      sum = sum + com(j + 1, i) * barray[i];
    barray[j] = (-1.0 * sum) / (j + 1);
  }
  for (j = 1; j <= (DATA_FILES + 1); j++)
  {
    if ((barray[j]< 0.0000001)&&(barray[j]> -0.0000001))
      barray[j] = 0.0;
  }
  return;
}
```

```
/********************************************************
** DATE: 01/16/1992                                    **
** VERSION: 1.1                                        **
** MODULE NAME: com()                                  **
** MODULE NUMBER: 1.4.8                                **
** DESCRIPTION:This module calculates the number of    **
**             possible  combinations                  **
** PASSED VARIABLES: n,i                               **
** RETURNS:com (double)                                **
** GLOBAL VARIABLES USED: None                         **
** GLOBAL VARIABLES CHANGED: None                      **
** FILES READ: None                                    **
** FILES WRITTEN: None                                 **
** HARDWARE INPUT: None                                **
** HARDWARE OUTPUT: None                               **
** MODULES CALT D: None                                **
** CALLING MODULES: Bern_Num(), Bern_Poly()            **
** AUTHOR: Salah A. Elewa                              **
** HISTORY:                                            **
**       1.0 Capt. K. N. Cole 08/05/1985               **
**           Translated from Pascal code               **
**           (Original by Capt. Mark Amell)            **
**       1.1 Salah A. Elewa  01/16/1992                **
**           Header modified, module name changed      **
********************************************************/
double com(int n, int i)
{
  double  prod, fn, fi, fj;

  fn = n;
  fi = i;
  f_ = fn - fi;
  prod = 1.0;
  while (fi > 0.0)
```

```
  {
    prod = prod * (fn / fi);
    fn = fn - 1.0;
    fi = fi - 1.0;
  }
  while (fj > 0.0)
  {
    prod = prod * (fn / fj);
    fn = fn - 1.0;
    fj = fj - 1.0;
  }
  return (prod);
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.0                                       **
** TITLE: Percentile Points File                      **
** FILENAME: table.c                                  **
** COORDINATOR: Salah Amin Elewa                      **
** PROJECT:Development of an Environment for Software **
**                Reliability Model Selection.        **
** OPERATING SYSTEM: MS DOS version 2.3 or higher     **
** LANGUAGE: Turbo C  (2.1)                           **
** FILE PROCESSING: Compile and link with files Prob.c**
**               and coefs.c                          **
** CONTENTS:                                          **
**          1.0   - main()                            **
**          1.0.1 - Get_Data()                        **
**          1.0.2 - Process_Equal()                   **
**          1.0.3 - Process-Unequal()                 **
**          1.0.4 - Print_Header()                    **
** FUNCTION: The modules in this file get information **
**     about calculation method, whether failures are**
**     equal or unequal, the number of samples, and  **
**     number of failures then a table of percentage **
**     points at various significance levels and var-**
**     ious total number of failures is then produced**
*******************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "defs.h"

extern void     Ar_Coef(double, double);
extern void     Ri_Coef(double, double, double, double);
extern double   Newton(double, double, double, double, int, double, double);
```

```
extern double   Ar[];
extern boolean  exact;
extern double   ka[];

static double   p;
static boolean  equal;
static int      k, RINI, RINC;
static FILE     *fp6;
static double   alpha_i[MAX_ALPHA] =
{   /* These values can be changed in the defs.h file */
#ifdef ALPHA1
  ALPHA1,
#endif
#ifdef ALPHA2
  ALPHA2,
#endif
#ifdef ALPHA3
  ALPHA3,
#endif
#ifdef ALPHA4
  ALPHA4,
#endif
#ifdef ALPHA5
  ALPHA5,
#endif
};
```

```
/*******************************************************
** DATE: 01/16/1991                                  **
** VERSION: 1.0                                      **
** MODULE NAME: main                                 **
** MODULE NUMBER: 1.0                                **
** DESCRIPTION: This main program in for percentage  **
**              points generation                    **
** PASSED VARIABLES: None                            **
** RETURNS: None                                     **
** HARDWARE INPUT: None                              **
** HARDWARE OUTPUT: None                             **
** MODULES CALLED: Get_Data()                        **
**                 Process_Equal()                   **
**                 Process_Unequal()                 **
** CALLING MODULES: None                             **
** AUTHOR: Salah Amin Elewa                          **
** HISTORY:                                          **
**       1.0 Salah Amin Elewa 01/16/1992             **
**           original version                        **
*******************************************************/
void main()
{
  void  Get_Data(void);
  void  Process_Equal(void);
  void  Process_Unequal(void);

  clrscr();
  Get_Data();
  if (equal)
    Process_Equal(),
  else
    Process_Unequal();
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.0                                       **
** MODULE NAME: Get_Data()                            **
** MODULE NUMBER: 1.0.1                               **
** DESCRIPTION: This module is used for obtaining inf-**
**      ormation about the number of failures in dif-**
**      ferent samples whether equal or unequal, the **
**      required accuracy of calculations whether    **
**      exact or asymptotic and the initial and inc- **
**      remental number of failures.                 **
** PASSED VARIABLES: None                             **
** RETURNS: None                                      **
** GLOBAL VARIABLES USED: exact, equal, p, RINI, RINC **
** GLOBAL VARIABLES CHANGED: exact, equal             **
** FILES READ: None                                   **
** FILES WRITTEN: None                                **
** HARDWARE INPUT: Keyboard                           **
** HARDWARE OUTPUT: Screen                            **
** MODULES CALLED: None                               **
** CALLING MODULES: The main program                  **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**      1.0 Salah A. Elewa 01/16/1992                 **
**          original version                          **
*******************************************************/
void Get_Data(void)
{
  char  ch;

  puts("\n\n\t Enter (E)qual (e) OR (U)nequal (u) Calculations ?");
  for (;;)
  {
    ch = getche();
```

```
      if (ch == 'u' || ch == 'U')
      {
        equal = FALSE;
        printf("nequal\n");
      }
      else if (ch == 'e' || ch == 'E')
      {
        equal = TRUE;
        printf("qual\n");
      }
      else
      {
        printf("\n\t\t\t Sorry!  %c  is Unknown Choice", ch);
        puts("\n\t\t\t Try Again OR Press Ctrl+Break to Quit\n\n\n");
      }
      if (ch == 'u' || ch == 'U' || ch == 'e' || ch == 'E')
        break;
    }
    puts("\n\n\t Enter (E)xact (e) OR (A)symptotic (a) Calculations ?");
    for (;;)
    {
      ch = getche();
      if (ch == 'a' || ch == 'A')
      {
        exact = FALSE;
        printf("symptotic\n");
      }
      else if (ch == 'e' || ch == 'E')
      {
        exact = TRUE;
        printf("xact\n");
      }
      else
      {
```

```
        printf("\n\t\t\t Sorry!   %c  is Unknown Choice", ch);
        puts("\n\t\t\t Try Again OR Press Ctrl+Break to Quit\n\n\n");
      }
      if (ch == 'a' || ch == 'A' || ch == 'e' || ch == 'E')
        break;
    }
    puts("\n\n\t Enter Number of Samples p?");
    scanf("%lf", &p);

    puts("\n\n\t Enter Initial Total Number of Failures RINI?");
    scanf("%d", &RINI);

    puts("\n\n\t Enter Increment to Total Number of Failures RINC?");
    scanf("%d", &RINC);
  }
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.0                                       **
** MODULE NAME: Process_Equal()                       **
** MODULE NUMBER: 1.0.2                               **
** DESCRIPTION: This module is used to calculate the  **
**       percentile points in case of equal number of **
**       failures in each sample.                     **
** PASSED VARIABLES: None                             **
** RETURNS: None                                      **
** GLOBAL VARIABLES USED: ka[],Ar[],p,RINI,RINC       **
**                        and Alpha_i[]               **
** GLOBAL VARIABLES CHANGED: ka[],Ar[]                **
** FILES READ: The file containing the failure times  **
**          or the times between failures             **
** FILES WRITTEN: "signific.dat" file, where the perc-**
**       entile points at different significance level**
**       and different failures per sample are printed**
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT: None                              **
** MODULES CALLED: Print_Header(),Ar_Coef(), Ri_Coef()**
**                 and Newton()                       **
** CALLING MODULES: The main program                  **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**       1.0 Salah A. Elewa 01/16/1992                **
**           original version                         **
*******************************************************/
void Process_Equal(void)
{
  void    Print_Header(void);
  int     i, r, R;
  double  xvalue, v, a, t, sum, delta, m;
```

```
if ((fp6 = fopen("signific.dat", "w")) == NULL)
{
  printf("can not open signific.dat file\n");
  exit(1);
}
delta = 0.0;
Print_Header();
for (i = 0; i < (int) p; i++)
  ka[i] = 1.0 / p;
v = 3.0 * (p - 1.0) / 2.0;
sum = 0.0;
for (i = 0; i < (int) p; i++)
  sum = sum + (1.0 / ka[i]);
delta = (13.0 * (sum - 1.0)) / ((p - 1.0) * 18.0);
a = (1.0 - v) / 2.0;
t = (p + 1.0) / (6.0 * p);
Ri_Coef(p, delta, a, v);
for (r = RINI; r <= RINI + 12 * RINC; r = r + RINC)
{                                    /* loop through r values */
  if (r < ((int) p + 1))
    r = (int) (p + 1.0);
  printf("\n|    r=%3d   |", r);
  fprintf(fp6, "\n|    r=%3d   |", r);
  R = r * (int) p;
  m = ((double) R - delta) / p;
  for (k = 0; k < MAX_ALPHA && alpha_i[k] > 0.0; k++)
  {
    xvalue = Newton(a, v, t, p, R, alpha_i[k], m);
    printf(" %8.6lf |", xvalue);
   fprintf(fp6, " %8.6lf |", xvalue);
  }
}
printf("\n└──────── ──────┴──────────┴──────────┴──────── ");
printf(" ┴──────────┴──────────┘\n");
```

```
  fprintf(fp6,"\n└──────────────┴─────────┴─────────┴");
  fprintf(fp6," ─────────┴───────┴─────────┘ \n");
  fclose(fp6);
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.0                                       **
** MODULE NAME: Process_Unequal()                     **
** MODULE NUMBER: 1.0.3                               **
** DESCRIPTION: This module is used to calculate the  **
**       percentile points in case of unequal number of*
**       failures in each sample.                     **
** PASSED VARIABLES: None                             **
** RETURNS: None                                      **
** GLOBAL VARIABLES USED: ka[],Ar[],p,RINI,RINC       **
**                        and Alpha_i[],k,MAX_ALPHA   **
** GLOBAL VARIABLES CHANGED: ka[], Ar[]               **
** FILES READ: The file "ka2" or "ka3" (depending on  **
**       the number of samples). These files contain  **
**       the values of the ka[] coefficients. Note that*
**       this module handles two cases of number of   **
**       samples but can easily modified (just redraw **
**       the header) to handle any number of samples. **
** FILES WRITTEN: "signific.dat" file, where the perc-**
**       entile points at different significance level**
**       and different values of ka[] are printed     **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT: None                              **
** MODULES CALLED: Print_Header(),Ar_Coef(), Ri_Coef()**
**                 and Newton()                       **
** CALLING MODULES: The main program                  **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**       1.0 Salah A. Elewa 01/16/1992                **
**           original version                         **
*******************************************************/
```

```
void Process_Unequal(void)
{
  FILE    *fp10;
  int     i, j, R;
  void    Print_Header(void);
  double  xvalue, v, a, t;
  double  sum, delta, m;

  delta = m = 0.0;
  if ((fp6 = fopen("signific.dat", "w")) == NULL)
  {
    printf("can not open signific.dat file\n");
    exit(1);
  }

  for (k = 0; k < MAX_ALPHA && alpha_i[k] > 0.0; k++)
  /* loop through alpha values */
  {
    Print_Header();
    if (p < 2.9999)
    {
      if ((fp10 = fopen("ka2", "r")) == NULL)
      {
       printf("\n Can NOT Find The File %s\n", fp10);
      puts("\n Enter the CORRECT File Name OR Ctrl-C to Exit\n");
      }
    }
    else
    {
      if ((fp10 = fopen("ka3", "r")) == NULL)
      {
       printf("\n\t\t\t Can NOT Find The File %s\n", fp10);
      puts("\n\t\t\t Enter the CORRECT File Name OR  Ctrl-C to Exit\n");
      }
```

```
}
while (!feof(fp10))
{
  if (p < 2.9999)
  {
   fscanf(fp10, "%lf %lf ", &ka[0], &ka[1]);
   printf("|%2.2f|%2.2lf|", ka[0], ka[1]);
   fprintf(fp6, "|%2.2f|%2.2lf|", ka[0], ka[1]);
  }
  else
  {
   fscanf(fp10, "%lf %lf %lf ", &ka[0], &ka[1], &ka[2]);
   printf("|%2.2f|%2.2lf|%2.2f|", ka[0], ka[1], ka[2]);
   fprintf(fp6, "|%2.2f|%2.2lf|%2.2f|", ka[0], ka[1], ka[2]);
  }
  v = 3.0 * (p - 1.0) / 2.0;
  sum = 0.0;
 for (i = 0; i < (MAX_SAMPLES + 1) && ka[i] > .0000001; i++)
   sum = sum + (1.0 / ka[i]);
 delta = (13.0 * (sum - 1.0)) / ((p - 1.0) * 18.0);
 a = (1.0 - v) / 2.0;
 t = (p + 1.0) / (6.0 * p);
 Ri_Coef(p, delta, a, v);
  for (j = 1; j <= 5; j++)
  {
   R = RINI + (j - 1) * RINC;
   m = ((double) R - delta) / p;
   if (ka[0] * R > 2.99999999)
    {
      xvalue = Newton(a, v, t, p, R, alpha_i[k], m);
      printf(" %8.6lf |", xvalue);
      fprintf(fp6, " %8.6lf |", xvalue);
   }
   else
```

```
      {
        printf("          |");
        fprintf(fp6, "          |");
      }
     }                              /* loop through fail values */
     printf("\n");
     fprintf(fp6, "\n");
   }
   if (p < 2.99999)
   {
     printf("└────┴────┴────────┴────────┴────────");
     printf("───┴────────┴────────┘\n");
     fprintf(fp6,"└────┴────┴────────┴────────┴──");
     fprintf(fp6,"───┴───────┴───────┘\n");

   }
   else
   {
     printf("└───┴───┴───┴────────┴────────┴────");
     printf("────┴────────┴────────┘\n");
     fprintf(fp6,"└───┴───┴───┴────────┴───────");
     fprintf(fp6,"─┴───────┴────────┴────────┘\n");
   }
   printf("\n\n\n");
   fprintf(fp6, "\n\n\n");
 }
 fclose(fp10);
 fclose(fp6);
}
```

```
/*******************************************************
** DATE: 01/16/1992                                   **
** VERSION: 1.0                                       **
** MODULE NAME: Print_Header()                        **
** MODULE NUMBER: 1.0.4                               **
** DESCRIPTION: This module is used for printing the  **
**       table header both on the screen and the file **
**       named signific.dat                           **
** PASSED VARIABLES: None                             **
** RETURNS: None                                      **
** GLOBAL VARIABLES USED: exact, equal, k, p, RINI,   **
**       RINC,alpha_i                                 **
** GLOBAL VARIABLES CHANGED: None                      **
** FILES READ: None                                   **
** FILES WRITTEN: "signific.dat" file                 **
** HARDWARE INPUT: None                               **
** HARDWARE OUTPUT: None                              **
** MODULES CALLED: None                               **
** CALLING MODULES: The main program                  **
** AUTHOR: Salah A. Elewa                             **
** HISTORY:                                           **
**       1.0 Salah A. Elewa 01/16/1992                **
**           original version                         **
*******************************************************/
void Print_Header(void)
{
 clrscr();
 if (equal)
 {
 printf("\n   Percentage Points of L=  ^(p/R) with Equal Sample Sizes");
 fprintf(fp6,"\n  Percentage Points of L=  ^(p/R) with Equal Sample Sizes");
 }
 else
 {
```

```
printf("\n   Percentage Points of L=  ^(p/R) with Unequal Sample Sizes");
fprintf(fp6, "\n   Percentage Points of L=  ^(p/R) with Unequal Sample
Sizes");
}

if (exact)
{
  printf("\n\t   Exact Method is used for Calculations");
  fprintf(fp6, "\n\t   Exact Method is used for Calculations");
}
else
{
  printf("\n\t   Asymptotic Method is used for Calculations");
  fprintf(fp6, "\n\t   Asymptotic Method is used for Calculations");
}
printf("\n\t\t   Number of samples = %d\n", (int) p);
fprintf(fp6, "\n\t\t   Number of samples = %d\n", (int) p);
if (equal)
{
  printf("┌───────────┬──────────────────────────────────────────────");
  printf("───────────┐\n");
  printf("│  Failures  │\t\t\t Level of Significance α'\t    │\n");
  printf("│           ├─────────┬──────────┬──────────┬───────────");
  printf("─┬──────────┤\n");
  printf("│ Per Sample │   0.100  │   0.050  │   0.025  │  0.010  ");
  printf(" │  0.005   │\n");
  printf("├───────────┼─────────┼──────────┼──────────┼───────────");
  printf("─┼──────────┤");

  fprintf(fp6,"┌──────────┬──────────────────────────────────────────");
  fprintf(fp6,"───────────────┐\n");
  fprintf(fp6,"│  Failures  │\t\t\t  Level of Significance α'\t    │\n");
  fprintf(fp6,"│          ├─────────┬──────────┬──────────┬───────────");
  fprintf(fp6,"───┬──────────┤\n");
```

```
  fprintf(fp6,"| Per Sample |   0.100   |   0.050   |   0.025   |  0.010  ");
  fprintf(fp6," |  0.005   |\n");
  fprintf(fp6,"├──────────┼──────────┼──────────┼──────────┼──────── ");
  fprintf(fp6," ──┼──────────┤");
}
else
{
  if (p < 2.9999)
  {
    printf(" ┌─────────┬──────────────────────────────────────────────── ");
    printf(" ─────────┐\n");
    printf("|'α= %5.3lf|\t\t  Total number of Failures R\t\t |\n",
          alpha_i[k]);
    printf("├───┬───┼───────────┬───────────┬─────────┬────────┬");
    printf("────────┤ \n");
    printf("| k1 | k2 |    %2d     |     %2d     |", RINI, RINI+RINC);
    printf("   %2d     |    %2d     |     %2d     |\n",RINI+2*RINC,
           RINI+3*RINC,RINI+4*RINC);
    printf("├───┼───┼───────────┼───────────┼─────────┼────────┼");
    printf("────────┤ \n");

    fprintf(fp6," ┌─────────┬──────────────────────────────────────────── ");
    fprintf(fp6," ──────────────┐ \n");
    fprintf(fp6, "|'α= %5.3lf|\t\t  Total number of Failures R\t\t |\n",
           alpha_i[k]);
    fprintf(fp6, "├───┬───┼───────────┬───────────┬─────────┬──────── ");
    fprintf(fp6," ──┬─────────┤ \n");
    fprintf(fp6, "| k1 | k2 |    %2d     |     %2d     |", RINI,RINI+RINC);
    fprintf(fp6, "  %2d     |    %2d     |     %2d     |\n",RINI+2*RINC,
           RINI+3*RINC,RINI+4*RINC);
    fprintf(fp6, "├───┼───┼───────────┼───────────┼─────────┼──────── ");
    fprintf(fp6,"─┼─────────┤ \n");
  }
  else
```

```
{
  printf("┌─────────┬────────────────────────────────────┬ ");
  printf("──────┐\n");
  printf("|  α = %5.3lf   |\t\t Total number of Failures R\t     |\n",
        alpha_i[k]);
  printf("├───┬───┬───┬───────┬───────┬──────┬─────┬ ");
  printf(" ──────┤\n");
  printf("| k1 | k2 | k3 |   %2d    |    %2d    |", RINI,RINI+RINC);
  printf("  %2d     |    %2d    |    %2d   |\n",RINI+2*RINC,
        RINI+3*RINC,RINI+4*RINC);
  printf("├───┼───┼───┼───────┼───────┼──────┼─────┼ ");
  printf("         \n");

  fprintf(fp6," ┌─────────┬──────────────────────────────────── ");
  fprintf(fp6," ─────────┐\n");
  fprintf(fp6, "|  α = %5.3lf   |\t\t Total number of Failures R\t     |\n",
        alpha_i[k]);
  fprintf(fp6, "├───┬───┬───┼───────┬───────┬──────┬──────── ");
  fprintf(fp6," ──┬──────┤ \n");
  fprintf(fp6,"| k1 | k2 | k3 |    %2d    |    %2d    |", RINI,
        RINI+RINC);
  fprintf(fp6,"  %2d     |    %2d    |    %2d    |\n",RINI+2*RINC,
        RINI+3*RINC,RINI+4*RINC);
  fprintf(fp6,"├───┼───┼───┼───────┼───────┼──────┼──── ");
  fprintf(fp6," ─┼──────┤ \n");
 }
}
}
```

Table D.1 Data Analysis of Failure Set 1

F. Set	*Significance Level α*							
	0.00025	0.0005	0.001	0.0025	0.005	.01	0.025	0.05
SET_1	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_2	Y/0.092 L=0.994 C=0.903	Y/0.084 L=0.994 C=0.910	Y/0.077 L=0.994 C=0.917	Y/0.068 L=0.994 C=0.926	Y/0.060 L=0.994 C=0.934	Y/0.053 L=0.994 C=0.941	Y/0.043 L=0.994 C=0.951	Y/0.035 L=0.994 C=0.959
SET_4	N/-0.01 L=0.895 C=0.902							
SET_5	N/-0.98 L=0.000 C=0.980							
SET_6	N/-0.23 L=0.683 C=0.911							
SET_14c	N/-0.85 L=0.040 C=0.892							
SET_17	N/-0.01 L=0.880 C=0.894							
SET_27	N/-0.68 L=0.211 C=0.895							
SET_S1a	N/-0.92 L=0.000 C=0.925							
SET_S1c	N/-0.95 L=0.000 C=0.954							
SET_SS3	N/-0.95 L=0.000 C=0.954							
SET_SS4	N/-0.94 L=0.000 C=0.943							
SET_Lt2	Y/0.011 L=0.927 C=0.916	Y/0.005 L=0.927 C=0.922	N/-0.00 L=0.927 C=0.929					
SET_Lt3	N/-0.08 L=0.867 C=0.945							

Table D.2 Data Analysis of Failure Set 2

F. Set	Significance Level α							
	0.00025	0.0005	0.001	0.0025	0.005	.01	0.025	0.05
SET_1	Y/0.092 L=0.994 C=0.903	Y/0.084 L=0.994 C=0.910	Y/0.077 L=0.994 C=0.917	Y/0.068 L=0.994 C=0.926	Y/0.060 L=0.994 C=0.934	Y/0.053 L=0.994 C=0.941	Y/0.043 L=0.994 C=0.951	Y/0.035 L=0.994 C=0.959
SET_2	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_4	N/-0.03 L=0.804 C=0.833							
SET_5	N/-0.28 L=0.700 C=0.978							
SET_6	N/-0.32 L=0.538 C=0.857							
SET_14c	N/-0.73 L=0.074 C=0.804							
SET_17	Y/0.074 L=0.881 C=0.808	Y/0.060 L=0.881 C=0.821	Y/0.047 L=0.881 C=0.834	Y/0.029 L=0.881 C=0.853	Y/0.015 L=0.881 C=0.867	N/-0.00 L=0.881 C=0.881		
SET_27	N/-0.53 L=0.282 C=0.813							
SET_S1a	N/-0.73 L=0.157 C=0.889							
SET_S1c	N/-0.54 L=0.405 C=0.943							
SET_SS3	N/-0.57 L=0.371 C=0.943							
SET_SS4	N/-0.70 L=0.226 C=0.925							
SET_Lt2	Y/0.092 L=0.962 C=0.870	Y/0.082 L=0.962 C=0.879	Y/0.073 L=0.962 C=0.889	Y/0.060 L=0.962 C=0.901	Y/0.051 L=0.962 C=0.911	Y/0.0412 L=0.9622 C=0.9210	Y/0.027 L=0.962 C=0.934	Y/0.017 L=0.962 C=0.945
SET_Lt3	N/-0.08 L=0.848 C=0.928							

Table D.3 Data Analysis of Failure Set 4

F. Set	*Significance Level α*							
	0.00025	0.0005	0.001	0.0025	0.005	.01	0.025	0.05
SET_1	N/-0.01 L=0.895 C=0.902							
SET_2	N/-0.03 L=0.804 C=0.833							
SET_4	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_5	N/-0.33 L=0.649 C=0.978							
SET_6	Y/0.030 L=0.886 C=0.856	Y/0.020 L=0.886 C=0.867	Y/0.010 L=0.886 C=0.877	N/-0.00 L=0.886 C=0.891				
SET_14c	N/-0.77 L=0.034 C=0.802							
SET_17	N/-0.25 L=0.557 C=0.806							
SET_27	N/-0.68 L=0.135 C=0.811							
SET_S1a	N/-0.78 L=0.104 C=0.889							
SET_S1c	N/-0.61 L=0.330 C=0.943							
SET_SS3	N/-0.64 L=0.303 C=0.943							
SET_SS4	N/-0.75 L=0.173 C=0.925							
SET_Lt2	N/-0.16 L=0.708 C=0.869							
SET_Lt3	Y/0.068 L=0.996 C=0.928	Y/0.063 L=0.996 C=0.933	Y/0.057 L=0.996 C=0.939	Y/0.050 L=0.996 C=0.946	Y/0.045 L=0.996 C=0.951	Y/0.039 L=0.996 C=0.957	Y/0.032 L=0.996 C=0.964	Y/0.026 L=0.996 C=0.970

Table D.4 Data Analysis of Failure Set 5

F. Set	*Significance Level α*							
	0.00025	0.0005	0.001	0.0025	0.005	.01	0.025	0.05
SET_1	N/-0.98 L=0.000 C=0.980							
SET_2	N/-0.28 L=0.700 C=0.978							
SET_4	N/-0.33 L=0.649 C=0.978							
SET_5	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_6	N/-0.46 L=0.517 C=0.979							
SET_14c	Y/0.007 L=0.984 C=0.978	Y/0.005 L=0.984 C=0.979	Y/0.003 L=0.984 C=0.981	Y/0.001 L=0.984 C=0.983	N/-0.00 L=0.984 C=0.985			
SET_17	N/-0.18 L=0.798 C=0.978							
SET_27	N/-0.08 L=0.899 C=0.978							
SET_S1a	Y/0.020 L=0.999 C=0.980	Y/0.018 L=0.999 C=0.981	Y/0.017 L=0.999 C=0.983	Y/0.015 L=0.999 C=0.985	Y/0.013 L=0.999 C=0.986	Y/0.011 L=0.999 C=0.988	Y/0.009 L=0.999 C=0.990	Y/0.008 L=0.999 C=0.992
SET_S1c	Y/0.012 L=0.995 C=0.983	Y/0.011 L=0.995 C=0.984	Y/0.009 L=0.995 C=0.985	Y/0.007 L=0.995 C=0.987	Y/0.006 L=0.995 C=0.988	Y/0.005 L=0.995 C=0.990	Y/0.003 L=0.995 C=0.992	Y/0.002 L=0.995 C=0.993
SET_SS3	Y/0.006 L=0.989 C=0.983	Y/0.005 L=0.989 C=0.984	Y/0.003 L=0.989 C=0.985	Y/0.002 L=0.989 C=0.987	Y/0.000 L=0.989 C=0.988	N/-0.00 L=0.989 C=0.990		
SET_SS4	N/-0.10 L=0.882 C=0.981							
SET_Lt2	N/-0.37 L=0.611 C=0.979							
SET_Lt3	N/-0.98 L=0.000 C=0.982							

Table D.5 Data Analysis of Failure Set 6

F. Set	Significance Level α							
	0.00025	0.00050	0.00100	0.00250	0.00500	0.01000	0.02500	0.05000
SET_1	N/-0.23 L=0.683 C=0.911							
SET_2	N/-0.32 L=0.538 C=0.857							
SET_4	Y/0.030 L=0.886 C=0.856	Y/0.020 L=0.886 C=0.867	Y/0.010 L=0.886 C=0.877	N/-0.00 L=0.886 C=0.891				
SET_5	N/-0.46 L=0.517 C=0.979							
SET_6	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_14c	N/-0.82 L=0.013 C=0.835							
SET_17	N/-0.51 L=0.325 C=0.838							
SET_27	N/-0.78 L=0.060 C=0.842							
SET_S1a	N/-0.86 L=0.043 C=0.900							
SET_S1c	N/-0.75 L=0.194 C=0.946							
SET_SS3	N/-0.77 L=0.174 C=0.946							
SET_SS4	N/-0.85 L=0.083 C=0.930							
SET_Lt2	N/-0.42 L=0.462 C=0.885							
SET_Lt3	N/-0.02 L=0.909 C=0.933							

Table D.6 Data Analysis of Failure Set 14C

F. Set	*Significance Level α*							
	0.00025	0.00050	0.00100	0.00250	0.00500	0.01000	0.02500	0.05000
SET_1	N/-0.85 L=0.040 C=0.892							
SET_2	N/-0.73 L=0.074 C=0.804							
SET_4	N/-0.77 L=0.034 C=0.802							
SET_5	Y/0.007 L=0.984 C=0.978	Y/0.005 L=0.984 C=0.979	Y/0.003 L=0.984 C=0.981	Y/0.001 L=0.984 C=0.983	N/-0.00 L=0.984 C=0.985			
SET_6	N/-0.82 L=0.013 C=0.835							
SET_14c	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_17	N/-0.62 L=0.145 C=0.766							
SET_27	N/-0.19 L=0.579 C=0.774							
SET_S1a	Y/0.040 L=0.916 C=0.876	Y/0.031 L=0.916 C=0.885	Y/0.022 L=0.916 C=0.894	Y/0.010 L=0.916 C=0.906	Y/0.001 L=0.916 C=0.915	N/-0.01 L=0.916 C=0.925		
SET_S1c	Y/0.032 L=0.972 C=0.939	Y/0.028 L=0.972 C=0.944	Y/0.023 L=0.972 C=0.948	Y/0.017 L=0.972 C=0.954	Y/0.013 L=0.972 C=0.959	Y/0.008 L=0.972 C=0.964	Y/0.002 L=0.972 C=0.970	N/-0.00 L=0.972 C=0.975
SET_SS3	N/-0.00 L=0.936 C=0.940							
SET_SS4	N/-0.09 L=0.828 C=0.919							
SET_Lt2	N/-0.77 L=0.081 C=0.851							
SET_Lt3	N/-0.92 L=0.000 C=0.923							

Table D.7 Data Analysis of Failure Set 17

F. Set	Significance Level α							
	0.00025	0.00050	0.00100	0.00250	0.00500	0.01000	0.02500	0.05000
SET_1	N/-0.01 L=0.880 C=0.894							
SET_2	Y/0.074 L=0.881 C=0.808	Y/0.060 L=0.881 C=0.821	Y/0.047 L=0.881 C=0.834	Y/0.029 L=0.881 C=0.853	Y/0.015 L=0.881 C=0.867	N/-0.00 L=0.881 C=0.881		
SET_4	N/-0.25 L=0.557 C=0.806							
SET_5	N/-0.18 L=0.798 C=0.978							
SET_6	N/-0.51 L=0.325 C=0.838							
SET_14c	N/-0.62 L=0.145 C=0.766							
SET_17	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_27	N/-0.33 L=0.448 C=0.779							
SET_S1a	N/-0.60 L=0.278 C=0.878							
SET_S1c	N/-0.38 L=0.556 C=0.940							
SET_SS3	N/-0.42 L=0.521 C=0.940							
SET_SS4	N/-0.56 L=0.364 C=0.920							
SET_Lt2	Y/0.108 L=0.962 C=0.854	Y/0.098 L=0.962 C=0.864	Y/0.087 L=0.962 C=0.875	Y/0.073 L=0.962 C=0.889	Y/0.062 L=0.962 C=0.900	Y/0.051 L=0.962 C=0.911	Y/0.036 L=0.962 C=0.926	Y/0.024 L=0.962 C=0.938
SET_Lt3	N/-0.27 L=0.649 C=0.923							

Table D.8 Data Analysis of Failure Set 27

F. Set	Significance Level α							
	0.00025	0.00050	0.00100	0.00250	0.00500	0.01000	0.02500	0.05000
SET_1	N/-0.68 L=0.211 C=0.895							
SET_2	N/-0.53 L=0.282 C=0.813							
SET_4	N/-0.68 L=0.135 C=0.811							
SET_5	N/-0.08 L=0.899 C=0.978							
SET_6	N/-0.78 L=0.060 C=0.842							
SET_14c	N/-0.19 L=0.579 C=0.774							
SET_17	N/-0.33 L=0.448 C=0.779							
SET_27	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_S1a	N/-0.32 L=0.558 C=0.880							
SET_S1c	N/-0.17 L=0.773 C=0.940							
SET_SS3	N/-0.22 L=0.722 C=0.941							
SET_SS4	N/-0.36 L=0.564 C=0.921							
SET_Lt2	N/-0.51 L=0.350 C=0.857							
SET_Lt3	N/-0.83 L=0.097 C=0.924							

Table D.9 Data Analysis of Failure Set SS1a

F. Set	*Significance Level α*							
	0.00025	0.00050	0.00100	0.00250	0.00500	0.01000	0.02500	0.05000
SET_1	N/-0.92 L=0.000 C=0.925							
SET_2	N/-0.73 L=0.157 C=0.889							
SET_4	N/-0.78 L=0.104 C=0.889							
SET_5	Y/0.020 L=0.999 C=0.980	Y/0.018 L=0.999 C=0.981	Y/0.017 L=0.999 C=0.983	Y/0.015 L=0.999 C=0.985	Y/0.013 L=0.999 C=0.986	Y/0.011 L=0.999 C=0.988	Y/0.009 L=0.999 C=0.990	Y/0.008 L=0.999 C=0.992
SET_6	N/-0.86 L=0.043 C=0.900							
SET_14c	Y/0.040 L=0.916 C=0.876	Y/0.031 L=0.916 C=0.885	Y/0.022 L=0.916 C=0.894	Y/0.010 L=0.916 C=0.906	Y/0.001 L=0.916 C=0.915	N/-0.01 L=0.916 C=0.925		
SET_17	N/-0.60 L=0.278 C=0.878							
SET_27	N/-0.32 L=0.558 C=0.880							
SET_S1a	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_S1c	Y/0.040 L=0.991 C=0.952	Y/0.036 L=0.991 C=0.955	Y/0.032 L=0.991 C=0.959	Y/0.027 L=0.991 C=0.964	Y/0.024 L=0.991 C=0.967	Y/0.020 L=0.991 C=0.971	Y/0.015 L=0.991 C=0.976	Y/0.011 L=0.991 C=0.980
SET_SS3	Y/0.040 L=0.992 C=0.952	Y/0.037 L=0.992 C=0.955	Y/0.033 L=0.992 C=0.959	Y/0.028 L=0.992 C=0.964	Y/0.025 L=0.992 C=0.967	Y/0.021 L=0.992 C=0.971	Y/0.016 L=0.992 C=0.976	Y/0.012 L=0.992 C=0.980
SET_SS4	N/-0.06 L=0.884 C=0.939							
SET_Lt2	N/-0.79 L=0.112 C=0.907							
SET_Lt3	N/-0.94 L=0.000 C=0.941							

Table D.10 Data Analysis of Failure Set SS1c

F. Set	Significance Level α							
	0.00025	0.00050	0.00100	0.00250	0.00500	0.01000	0.02500	0.05000
1	N/-0.95 L=0.000 C=0.954							
2	N/-0.54 L=0.405 C=0.943							
4	N/-0.61 L=0.330 C=0.943							
5	Y/0.012 L=0.995 C=0.983	Y/0.011 L=0.995 C=0.984	Y/0.009 L=0.995 C=0.985	Y/0.007 L=0.995 C=0.987	Y/0.006 L=0.995 C=0.988	Y/0.005 L=0.995 C=0.990	Y/0.003 L=0.995 C=0.992	Y/0.002 L=0.995 C=0.993
6	N/-0.75 L=0.194 C=0.946							
14C	Y/0.032 L=0.972 C=0.939	Y/0.028 L=0.972 C=0.944	Y/0.023 L=0.972 C=0.948	Y/0.017 L=0.972 C=0.954	Y/0.013 L=0.972 C=0.959	Y/0.008 L=0.972 C=0.964	Y/0.002 L=0.972 C=0.970	N/-0.00 L=0.972 C=0.975
17	N/-0.38 L=0.556 C=0.940							
27	N/-0.17 L=0.773 C=0.940							
SS1a	Y/0.040 L=0.991 C=0.952	Y/0.0364 L=0.9911 C=0.9557	Y/0.032 L=0.991 C=0.959	Y/0.027 L=0.991 C=0.964	Y/0.024 L=0.991 C=0.967	Y/0.0206 L=0.9911 C=0.9715	Y/0.015 L=0.991 C=0.976	Y/0.011 L=0.991 C=0.980
ss1c	Two Identical Sets or Set Compared with Itself L = 1.0							
SS3	N/-0.00 L=0.964 C=0.966							
SS4	N/-0.16 L=0.799 C=0.960							
Litt2	N/-0.64 L=0.310 C=0.948							
Litt3	N/-0.96 L=0.000 C=0.961							

Table D.11 Data Analysis of Failure Set SS3

F. Set	*Significance Level α*							
	0.00025	0.0005	0.001	0.0025	0.005	.01	0.025	0.05
SET_1	N/-0.95 L=0.000 C=0.954							
SET_2	N/-0.57 L=0.371 C=0.943							
SET_4	N/-0.64 L=0.303 C=0.943							
SET_5	Y/0.006 L=0.989 C=0.983	Y/0.005 L=0.989 C=0.984	Y/0.003 L=0.989 C=0.985	Y/0.002 L=0.989 C=0.987	Y/0.000 L=0.989 C=0.988	N/-0.00 L=0.989 C=0.990		
SET_6	N/-0.77 L=0.174 C=0.946							
SET_14c	N/-0.00 L=0.936 C=0.940							
SET_17	N/-0.42 L=0.521 C=0.940							
SET_27	N/-0.22 L=0.722 C=0.941							
SET_S1a	Y/0.040 L=0.992 C=0.952	Y/0.037 L=0.992 C=0.955	Y/0.033 L=0.992 C=0.959	Y/0.028 L=0.992 C=0.964	Y/0.025 L=0.992 C=0.967	Y/0.021 L=0.992 C=0.971	Y/0.016 L=0.992 C=0.976	Y/0.012 L=0.992 C=0.980
SET_S1c	N/-0.00 L=0.964 C=0.966							
SET_SS3	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_SS4	N/-0.04 L=0.920 C=0.960							
SET_Lt2	N/-0.67 L=0.274 C=0.948							
SET_Lt3	N/-0.96 L=0.000 C=0.961							

Table D.12 Data Analysis of Failure Set SS4

F. Set	Significance Level α							
	0.00025	0.00050	0.00100	0.00250	0.00500	0.01000	0.02500	0.05000
SET_1	N/-0.94 L=0.000 C=0.943							
SET_2	N/-0.70 L=0.226 C=0.925							
SET_4	N/-0.75 L=0.173 C=0.925							
SET_5	N/-0.10 L=0.882 C=0.981							
SET_6	N/-0.85 L=0.083 C=0.930							
SET_14c	N/-0.09 L=0.828 C=0.919							
SET_17	N/-0.56 L=0.364 C=0.920							
SET_27	N/-0.36 L=0.564 C=0.921							
SET_S1a	N/-0.06 L=0.884 C=0.939							
SET_S1c	N/-0.16 L=0.799 C=0.960							
SET_SS3	N/-0.04 L=0.920 C=0.960							
SET_SS4	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_Lt2	N/-0.78 L=0.149 C=0.934							
SET_Lt3	N/-0.95 L=0.000 C=0.953							

Table D.13 Data Analysis of Failure Set Litt2

F. Set	*Significance Level α*							
	0.00025	0.00050	0.00100	0.00250	0.00500	0.01000	0.02500	0.05000
SET_1	Y/0.011 L=0.927 C=0.916	Y/0.005 L=0.927 C=0.922	N/-0.00 L=0.927 C=0.929					
SET_2	Y/0.092 L=0.962 C=0.870	Y/0.082 L=0.962 C=0.879	Y/0.073 L=0.962 C=0.889	Y/0.060 L=0.962 C=0.901	Y/0.051 L=0.962 C=0.911	Y/0.041 L=0.962 C=0.921	Y/0.027 L=0.962 C=0.934	Y/0.017 L=0.962 C=0.945
SET_4	N/-0.16 L=0.708 C=0.869							
SET_5	N/-0.37 L=0.611 C=0.979							
SET_6	N/-0.42 L=0.462 C=0.885							
SET_14c	N/-0.77 L=0.081 C=0.851							
SET_17	Y/0.108 L=0.962 C=0.854	Y/0.098 L=0.962 C=0.864	Y/0.087 L=0.962 C=0.875	Y/0.073 L=0.962 C=0.889	Y/0.062 L=0.962 C=0.900	Y/0.051 L=0.962 C=0.911	Y/0.036 L=0.962 C=0.926	Y/0.024 L=0.962 C=0.938
SET_27	N/-0.51 L=0.350 C=0.857							
SET_S1a	N/-0.79 L=0.112 C=0.907							
SET_S1c	N/-0.64 L=0.310 C=0.948							
SET_SS3	N/-0.67 L=0.274 C=0.948							
SET_SS4	N/-0.78 L=0.149 C=0.934							
SET_Lt2	Two Identical Sets or Set Compared with Itself L = 1.0							
SET_Lt3	N/-0.27 L=0.671 C=0.936							

Table D.14 Data Analysis of Failure Set Litt3

F. Data	Significance Level α							
	0.00025	0.00050	0.00100	0.00250	0.00500	0.01000	0.02500	0.05000
SET_1	N/-0.08 L=0.867 C=0.945							
SET_2	N/-0.08 L=0.848 C=0.928							
SET_4	Y/0.068 L=0.996 C=0.928	Y/0.063 L=0.996 C=0.933	Y/0.057 L=0.996 C=0.939	Y/0.050 L=0.996 C=0.946	Y/0.045 L=0.996 C=0.951	Y/0.039 L=0.996 C=0.957	Y/0.032 L=0.996 C=0.964	Y/0.026 L=0.996 C=0.970
SET_5	N/-0.98 L=0.000 C=0.982							
SET_6	N/-0.02 L=0.909 C=0.933							
SET_14c	N/-0.92 L=0.000 C=0.923							
SET_17	N/-0.27 L=0.649 C=0.923							
SET_27	N/-0.83 L=0.097 C=0.924							
SET_S1a	N/-0.94 L=0.000 C=0.941							
SET_S1c	N/-0.96 L=0.000 C=0.961							
SET_SS3	N/-0.96 L=0.000 C=0.961							
SET_SS4	N/-0.95 L=0.000 C=0.953							
SET_Lt2	N/-0.27 L=0.671 C=0.936							
SET_Lt3	Two Identical Sets or Set Compared with Itself L = 1.0							

APPENDIX E. *Exact and Asymptotic Percentage Points of the Test Statistic*

Table E.1 Percentage Points of $L = \Lambda^{\frac{2}{r}R}$ when p=2

Failures	*Level of Significance* α				
per sample	0.100	0.050	0.025	0.010	0.005
r= 5	0.449752	0.368224	0.302557	0.234197	0.193293
r= 6	0.529323	0.451446	0.386168	0.315083	0.270573
r= 7	0.589488	0.516492	0.453664	0.383167	0.337681
r= 8	0.636320	0.568307	0.508641	0.440245	0.395162
r= 9	0.673746	0.610380	0.554012	0.488355	0.444380
r= 10	0.704265	0.645142	0.591966	0.529252	0.486724
r= 11	0.729615	0.674304	0.624117	0.564338	0.523397
r= 12	0.750998	0.699097	0.651667	0.594711	0.555387
r= 13	0.769267	0.720420	0.675515	0.621226	0.583491
r= 14	0.785056	0.738947	0.696349	0.644554	0.608347
r= 15	0.798822	0.755189	0.714698	0.665224	0.630470
r= 20	0.847698	0.813374	0.781060	0.740910	0.712234
r= 25	0.877481	0.849270	0.822471	0.788849	0.764602
r= 30	0.897547	0.873605	0.850742	0.821874	0.800924
r= 35	0.911960	0.891182	0.871259	0.845990	0.827569
r= 40	0.922823	0.904472	0.886824	0.864365	0.847940
r= 45	0.931300	0.914870	0.899034	0.878829	0.864014
r= 50	0.938100	0.923229	0.908868	0.890509	0.877020
r= 55	0.943676	0.930093	0.916958	0.900137	0.887758
r= 60	0.948330	0.935831	0.923729	0.908210	0.896773
r= 65	0.952274	0.940699	0.929480	0.915076	0.904449
r= 70	0.955658	0.944880	0.934425	0.920988	0.911063
r= 75	0.958595	0.948511	0.938722	0.926130	0.916822
r= 80	0.961167	0.951693	0.942490	0.930644	0.921880
r= 85	0.963438	0.954505	0.945822	0.934638	0.926359
r= 90	0.965458	0.957007	0.948789	0.938197	0.930352
r= 95	0.967266	0.959249	0.951448	0.941389	0.933935
r=100	0.968895	0.961268	0.953845	0.944268	0.937167
r=105	0.970369	0.963097	0.956016	0.946877	0.940098
r=110	0.971710	0.964761	0.957992	0.949252	0.942767
r=115	0.972935	0.966281	0.959799	0.951425	0.945209
r=120	0.974058	0.967676	0.961456	0.953419	0.947451
r=125	0.975091	0.968960	0.962982	0.955256	0.949517
r=130	0.976046	0.970145	0.964392	0.956953	0.951427
r=135	0.976930	0.971244	0.965698	0.958527	0.953197
r=140	0.954843	0.959989	0.966912	0.959989	0.954843
r=145	0.956377	0.961352	0.968043	0.961352	0.956377
r=150	0.957811	0.962625	0.969099	0.962625	0.957811

Table E.2 Percentage Points of $L = \Lambda^{\frac{2}{k}}$ when p=3

Failures	*Level of Significance α*				
per sample	0.100	0.050	0.025	0.010	0.005
r= 5	0.267091	0.209072	0.165154	0.121904	0.097222
r= 6	0.347988	0.286615	0.238030	0.187729	0.157509
r= 7	0.414579	0.352800	0.302399	0.248465	0.214986
r= 8	0.469791	0.409111	0.358512	0.303058	0.267820
r= 9	0.516029	0.457190	0.407306	0.351634	0.315624
r= 10	0.555170	0.498503	0.449832	0.394734	0.358595
r= 11	0.588651	0.534266	0.487063	0.433010	0.397157
r= 12	0.617573	0.565457	0.519835	0.467098	0.431792
r= 13	0.642778	0.592859	0.548847	0.497567	0.462971
r= 14	0.664924	0.617098	0.574674	0.524914	0.491123
r= 15	0.684524	0.638674	0.597791	0.549561	0.516626
= 20	0.756144	0.718456	0.684247	0.643088	0.614437
r= 25	0.801400	0.769596	0.740426	0.704924	0.679931
r= 30	0.832536	0.805089	0.779744	0.748663	0.726621
r= 35	0.855252	0.831139	0.808765	0.781180	0.761515
r= 40	0.872551	0.851062	0.831051	0.806281	0.788555
r= 45	0.886162	0.866788	0.848697	0.826235	0.810112
r= 50	0.897149	0.879515	0.863012	0.842472	0.827694
r= 55	0.906203	0.890025	0.874856	0.855940	0.842304
r= 60	0.913794	0.898850	0.884818	0.867290	0.854635
r= 65	0.920248	0.906365	0.893313	0.876986	0.865181
r= 70	0.925804	0.912841	0.900642	0.885362	0.874303
r= 75	0.930637	0.918480	0.907029	0.892672	0.882270
r= 80	0.934878	0.923435	0.912645	0.899107	0.889289
r= 85	0.938631	0.927821	0.917622	0.904814	0.895518
r= 90	0.941976	0.931733	0.922063	0.909911	0.901085
r= 95	0.944974	0.935242	0.926050	0.914490	0.906089
r=100	0.947678	0.938409	0.929648	0.918626	0.910611
r=105	0.950129	0.941280	0.932913	0.922381	0.914719
r=110	0.952360	0.943896	0.935889	0.925805	0.918465
r=115	0.954401	0.946288	0.938612	0.928939	0.921897
r=120	0.956274	0.948485	0.941113	0.931820	0.925051
r=125	0.957999	0.950509	0.943418	0.934476	0.927961
r=130	0.959593	0.952381	0.945550	0.936933	0.930653
r=135	0.961070	0.954116	0.947526	0.939213	0.933151
r=140	0.962444	0.955729	0.949365	0.941333	0.935476
r=145	0.963723	0.957232	0.951079	0.943311	0.937644
r=150	0.964919	0.958637	0.952681	0.945160	0.939672

Table E.3 Percentage Points of $L = \Lambda^{\frac{2}{R}}$ when p=4

Failures	*Level of Significance* α				
per sample	0.100	0.050	0.025	0.010	0.005
r= 5	0.163267	0.122321	0.092616	0.064596	0.049267
r= 6	0.236473	0.189151	0.153010	0.116830	0.095732
r= 7	0.300401	0.249806	0.209863	0.168396	0.143314
r= 8	0.356022	0.304056	0.262035	0.217265	0.189488
r= 9	0.404423	0.352264	0.309309	0.262631	0.233110
r= 10	0.446668	0.395040	0.351903	0.304292	0.273724
r= 11	0.483713	0.433048	0.390219	0.342349	0.311238
r= 12	0.516370	0.466919	0.424713	0.377045	0.345751
r= 13	0.545319	0.497216	0.455828	0.408675	0.377457
r= 14	0.571123	0.524427	0.483974	0.437543	0.406581
r= 15	0.594244	0.548967	0.509513	0.463938	0.433360
r= 20	0.680921	0.642223	0.607815	0.567186	0.539342
r= 25	0.737413	0.703997	0.673932	0.637973	0.613020
r= 30	0.777026	0.747748	0.721202	0.689182	0.666780
r= 35	0.806303	0.780303	0.756601	0.727839	0.707600
r= 40	0.828807	0.805451	0.784071	0.758012	0.739596
r= 45	0.846639	0.825451	0.805995	0.782199	0.765327
r= 50	0.861114	0.841734	0.823892	0.802011	0.786456
r= 55	0.873096	0.855244	0.838775	0.818533	0.804111
r= 60	0.883178	0.866633	0.851344	0.832517	0.819080
r= 65	0.891778	0.876363	0.862099	0.844506	0.831930
r= 70	0.899199	0.884772	0.871405	0.854896	0.843081
r= 75	0.905669	0.892111	0.879536	0.863988	0.852848
r= 80	0.911359	0.898572	0.886701	0.872009	0.861472
r= 85	0.916402	0.904304	0.893063	0.879139	0.869143
r= 90	0.920903	0.909423	0.898749	0.885516	0.876010
r= 95	0.924943	0.914022	0.903862	0.891256	0.882194
r=100	0.928592	0.918178	0.908483	0.896448	0.887790
r=105	0.931902	0.921950	0.912681	0.901167	0.892880
r=110	0.934919	0.925390	0.916511	0.905475	0.897528
r=115	0.937680	0.928540	0.920019	0.909424	0.901791
r=120	0.940217	0.931434	0.923244	0.913056	0.905713
r=125	0.942555	0.934104	0.926220	0.916408	0.909334
r=130	0.944717	0.936573	0.928973	0.919511	0.912687
r=135	0.946722	0.938864	0.931528	0.922393	0.915802
r=140	0.948587	0.940996	0.933907	0.925075	0.918702
r=145	0.950326	0.942984	0.936125	0.927578	0.921409
r=150	0.951951	0.944842	0.938199	0.929920	0.923941

Table E.4 Percentage Points of $L = \Lambda^{\frac{2}{r}}$ when p=5

Failures per sample	*Level of Significance* α				
	0.100	0.050	0.025	0.010	0.005
r= 6	0.163811	0.128060	0.101629	0.076017	0.061524
r= 7	0.220897	0.180198	0.148926	0.117309	0.098639
r= 8	0.273189	0.229512	0.195032	0.159134	0.137313
r= 9	0.320432	0.275122	0.238621	0.199779	0.175660
r= 10	0.362879	0.316854	0.279183	0.238406	0.212663
r= 11	0.400973	0.354853	0.316617	0.274657	0.247814
r= 12	0.435198	0.389399	0.351028	0.308440	0.586766
r= 13	0.466021	0.420820	0.382614	0.695443	0.491269
r= 14	0.493864	0.449441	0.411610	0.503296	0.340773
r= 15	0.519099	0.475566	0.438254	0.407376	0.367715
r= 20	0.616001	0.577404	0.543581	0.504181	0.477486
r= 25	0.681000	0.646942	0.616708	-0.580990	0.556462
r= 30	0.727395	0.697127	0.670026	0.637715	0.615329
r= 35	0.762100	0.734949	0.710492	0.681142	0.660680
r= 40	0.789012	0.764436	0.742200	0.715385	0.696602
r= 45	0.810477	0.788053	0.767692	0.743047	0.725723
r= 50	0.827992	0.807385	0.788623	0.765844	0.749786
r= 55	0.842552	0.823497	0.806109	0.784947	0.769994
r= 60	0.854845	0.837129	0.820933	0.801183	0.787199
r= 65	0.865360	0.848812	0.833659	0.815148	0.802020
r= 70	0.874458	0.858934	0.844700	0.827286	0.814919
r= 75	0.882405	0.867788	0.854369	0.837932	0.826245
r= 80	0.889408	0.875598	0.862908	0.847346	0.836269
r= 85	0.895624	0.882538	0.870502	0.855728	0.845203
r= 90	0.901179	0.888745	0.877300	0.863240	0.853214
r= 95	0.906173	0.894330	0.883421	0.870009	0.860438
r=100	0.910687	0.899382	0.888961	0.876141	0.866986
r=105	0.914787	0.903973	0.893999	0.881721	0.872948
r=110	0.918527	0.908164	0.898601	0.886820	0.878400
r=115	0.921953	0.912004	0.902819	0.891499	0.883403
r=120	0.925103	0.915537	0.906701	0.895807	0.888011
r=125	0.928008	0.918797	0.910285	0.899786	0.892269
r=130	0.930697	0.921814	0.913604	0.903472	0.896216
r=135	0.933192	0.924616	0.916687	0.906897	0.899884
r=140	0.935514	0.927224	0.919557	0.910088	0.903302
r=145	0.937679	0.929658	0.922236	0.913067	0.906494
r=150	0.939704	0.931934	0.924742	0.915855	0.909483

Table E.5 Percentage Points of $L = \Lambda^{\frac{2}{R}}$ with unequal sample sizes ($\alpha = .01,\ .05, p = 2$)

$\alpha = .01$		Total Number of Failures R				
k_1	k_2	10	15	20	25	30
.50	.50	0.234197	0.412950	0.529252	0.608405	0.665224
.45	.55	0.233263	0.412267	0.528772	0.608058	0.664963
.40	.60	0.230377	0.410141	0.527275	0.606973	0.664148
.35	.65	0.225282	0.406314	0.524565	0.605006	0.662670
.30	.70	0.217567	0.400287	0.520250	0.601861	0.660302
.25	.75		0.391187	0.513586	0.596967	0.656606
.20	.80		0.377726	0.503170	0.589184	0.650688
.15	.85			0.486527	0.576075	0.640521

$\alpha = .01$		Total Number of Failures R				
k_1	k_2	40	60	80	100	120
.50	.50	0.740910	0.821874	0.864365	0.890509	0.908210
.45	.55	0.740749	0.821796	0.864320	0.890479	0.908189
.40	.60	0.740246	0.821552	0.864176	0.890385	0.908123
.35	.65	0.739332	0.821108	0.863916	0.890215	0.908002
.30	.70	0.737868	0.820398	0.863499	0.889941	0.907809
.25	.75	0.735578	0.819285	0.862847	0.889513	0.907508
.20	.80	0.731893	0.817493	0.861796	0.888825	0.907022
.15	.85	0.725477	0.814358	0.859958	0.887621	0.906174

$\alpha = .05$		Total Number of Failures R				
k_1	k_2	10	15	20	25	30
.50	.50	0.368224	0.543806	0.645142	0.710145	0.755189
.45	.55	0.367196	0.543173	0.644733	0.709863	0.754983
.40	.60	0.364009	0.541197	0.643453	0.708979	0.754339
.35	.65	0.358338	0.537626	0.641132	0.707373	0.753169
.30	.70	0.349618	0.531955	0.637418	0.704798	0.751291
.25	.75		0.523260	0.631635	0.700771	0.748349
.20	.80		0.509963	0.622454	0.694305	0.743607
.15	.85			0.607198	0.683191	0.735355

$\alpha = .05$		Total Number of Failures R				
k_1	k_2	40	60	80	100	120
.50	.50	0.813374	0.873605	0.904472	0.923229	0.935831
.45	.55	0.813252	0.873548	0.904439	0.923207	0.935816
.40	.60	0.812868	0.873368	0.904335	0.923140	0.935769
.35	.65	0.812172	0.873042	0.904147	0.923018	0.935683
.30	.70	0.811054	0.872518	0.903845	0.922822	0.935546
.25	.75	0.809301	0.871698	0.903372	0.922515	0.935331
.20	.80	0.806470	0.870373	0.902610	0.922021	0.934985
.15	.85	0.801506	0.868049	0.901274	0.921156	0.934380

Table E.6 Percentage Points of $L = \Lambda^{\frac{2}{R}}$ with unequal sample sizes ($\alpha = .01,\ .05, p = 3$)

$\alpha = .01$			Total Number of Failures R				
k_1	k_2	k_3	10	15	20	25	30
.33	.33	.34	0.018296	0.121895	0.228876	0.319888	0.394727
.30	.30	.40	0.017713	0.121021	0.228008	0.319113	0.394061
.30	.35	.35	0.018151	0.121678	0.228652	0.319684	0.394549
.25	.25	.50		0.116438	0.223478	0.315090	0.390617
.25	.30	.45		0.119065	0.226019	0.317317	0.392509
.25	.35	.40		0.120352	0.227225	0.318355	0.393380
.20	.20	.60		0.107657	0.214517	0.307031	0.383685
.20	.30	.50		0.116144	0.222636	0.314076	0.389622

$\alpha = .01$			Total Number of Failures R				
k_1	k_2	k_3	40	50	60	70	80
.33	.33	.34	0.507002	0.585586	0.643085	0.686799	0.721083
.30	.30	.40	0.506519	0.585229	0.642813	0.686586	0.720912
.30	.35	.35	0.506871	0.585488	0.643011	0.686741	0.721036
.25	.25	.50	0.504038	0.583401	0.641424	0.685498	0.720040
.25	.30	.45	0.505389	0.584391	0.642174	0.686085	0.720510
.25	.35	.40	0.506004	0.584840	0.642514	0.686349	0.720721
.20	.20	.60	0.499028	0.579710	0.638619	0.683305	0.718282
.20	.30	.50	0.503225	0.582767	0.640926	0.685101	0.719717

$\alpha = .05$			Total Number of Failures R				
k_1	k_2	k_3	10	15	20	25	30
.33	.33	.34	0.057163	0.209061	0.331908	0.425966	0.498496
.30	.30	.40	0.056142	0.207997	0.330967	0.425188	0.497862
.30	.35	.35	0.056926	0.208798	0.331664	0.425760	0.498327
.25	.25	.50		0.202384	0.326053	0.421151	0.494584
.25	.30	.45		0.205611	0.328806	0.423383	0.496382
.25	.35	.40		0.207191	0.330107	0.424419	0.497209
.20	.20	.60		0.191514	0.316255	0.413024	0.487961
.20	.30	.50		0.202037	0.325075	0.420091	0.493610

$\alpha = .05$			Total Number of Failures R				
k_1	k_2	k_3	40	50	60	70	80
.33	.33	.34	0.601258	0.669779	0.718453	0.754727	0.782772
.30	.30	.40	0.600828	0.669472	0.718225	0.754551	0.782632
.30	.35	.35	0.601142	0.669695	0.718391	0.754679	0.782734
.25	.25	.50	0.598613	0.667900	0.717059	0.753654	0.781922
.25	.30	.45	0.599818	0.668752	0.717688	0.754138	0.782304
.25	.35	.40	0.600366	0.669137	0.717973	0.754355	0.782476
.20	.20	.60	0.594132	0 664723	0.714703	0.751843	0.780489
.20	.30	.50	0.597876	0.667350	0.716638	0.753324	0.781657

Table E.7 Comparison of Exact and Asymptotic Values of the Percentage Points for $L = \Lambda^{\frac{2}{n}}$ and $\alpha = 0.01$

Failures per Sample (r)	p=2			p=3			p=4		
	Exact Using (6)	First 2 terms of (19)	First term of (19)	Exact Using (6)	First 2 terms of (19)	First term of (19)	Exact Using (6)	First 2 terms of (19)	First term of (19)
5	0.234197	0.232728	0.231253	0.121904	0.113431	0.108148	0.064596	0.050114	0.044613
6	0.315083	0.314095	0.312921	0.187729	0.180706	0.175512	0.116830	0.102762	0.095544
7	0.383167	0.382519	0.381605	0.248465	0.243165	0.238549	0.168396	0.156777	0.149340
8	0.440245	0.439814	0.439103	0.303058	0.299160	0.295212	0.217265	0.208152	0.201178
9	0.488355	0.488062	0.487503	0.351634	0.348767	0.345437	0.262631	0.255590	0.249319
10	0.529252	0.529047	0.528601	0.394734	0.392605	0.389805	0.304292	0.298853	0.293321
11	0.564338	0.564191	0.563832	0.433010	0.431408	0.429048	0.342349	0.338121	0.333282
12	0.594711	0.594603	0.594310	0.467098	0.465873	0.463876	0.377045	0.373728	0.369507
13	0.621226	0.621146	0.620904	0.497567	0.496618	0.494918	0.408675	0.406046	0.402362
14	0.644554	0.644493	0.644292	0.524914	0.524168	0.522713	0.437543	0.435438	0.432216
15	0.665224	0.665176	0.665007	0.549561	0.548968	0.547715	0.463938	0.462236	0.459409
20	0.740910	0.740894	0.740813	0.643088	0.642867	0.642222	0.567186	0.566514	0.564964
25	0.788849	0.788842	0.788798	0.704924	0.704823	0.704452	0.637973	0.637658	0.636731
30	0.821874	0.821871	0.821844	0.748663	0.748611	0.748379	0.689182	0.689016	0.688421
35	0.845990	0.845988	0.845970	0.781180	0.781150	0.780996	0.7278396	0.727743	0.727340
40	0.864365	0.864364	0.864352	0.806281	0.806263	0.806155	0.758012	0.757952	0.757667
45	0.878829	0.878829	0.878820	0.826235	0.826223	0.826145	0.782199	0.782160	0.781951
50	0.890509	0.890509	0.890503	0.842472	0.842464	0.842405	0.802011	0.801985	0.801827
55	0.900137	0.900137	0.900132	0.855940	0.855934	0.855889	0.818533	0.818514	0.818392
60	0.908210	0.908210	0.908206	0.867290	0.867286	0.867251	0.832517	0.832504	0.832407
65	0.915076	0.915076	0.915073	0.876986	0.876983	0.876955	0.844506	0.844496	0.844419
70	0.920988	0.920987	0.920985	0.885362	0.885360	0.885337	0.854896	0.854889	0.854826
75	0.926130	0.926130	0.926128	0.892672	0.892671	0.892652	0.863988	0.863982	0.863930
80	0.930644	0.930644	0.930642	0.899107	0.899106	0.899090	0.872009	0.872005	0.871961
85	0.934638	0.934638	0.934636	0.904814	0.904813	0.904800	0.879139	0.879135	0.879098
90	0.938197	0.938197	0.938196	0.909911	0.909910	0.909899	0.885516	0.885513	0.885482
95	0.941389	0.941389	0.941388	0.914490	0.914489	0.914480	0.891256	0.891253	0.891227
100	0.944268	0.944268	0.944267	0.918626	0.918625	0.918617	0.896448	0.896446	0.896423
105	0.946877	0.946877	0.946876	0.922381	0.922380	0.922373	0.901167	0.901165	0.901145
110	0.949252	0.949252	0.949252	0.925805	0.925804	0.925798	0.905475	0.905474	0.905456
115	0.951425	0.951425	0.951424	0.928939	0.928939	0.928933	0.909424	0.909422	0.909407
120	0.953419	0.953419	0.953418	0.931820	0.931819	0.931815	0.913056	0.913055	0.913041
125	0.955256	0.955256	0.955255	0.934476	0.934476	0.934471	0.916408	0.916407	0.916395
130	0.956953	0.956953	0.956953	0.936933	0.936933	0.936929	0.919511	0.919511	0.919500
135	0.958527	0.958527	0.958526	0.939213	0.939212	0.939209	0.922393	0.922392	0.922382
140	0.959989	0.959989	0.959989	0.941333	0.941333	0.941330	0.925075	0.925075	0.925066

Table E.8 Comparison of Exact and Asymptotic Values of the Percentage Points for $L = \Lambda^{\frac{2}{n}}$ and $\alpha = 0.05$

Failures per Sample (r)	p=2			p=3			p=4		
	Exact Using (6)	First 2 terms of (19)	First term of (19)	Exact Using (6)	First 2 terms of (19)	First term of (19)	Exact Using (6)	First 2 terms of (19)	First term of (19)
5	0.368224	0.367279	0.365570	0.209072	0.200361	0.192507	0.122321	0.103594	09337113
6	0.451446	0.450909	0.449732	0.286615	0.280641	0.274142	0.189151	0.174706	16413844
7	0.516492	0.516177	0.515346	0.352800	0.348796	0.343631	0.249806	0.239354	22987540
8	0.568307	0.568113	0.567511	0.409111	0.406393	0.402313	0.304056	0.296553	28845278
9	0.610380	0.610256	0.609807	0.457190	0.455303	0.452062	0.352264	0.346825	34001619
10	0.645142	0.645059	0.644717	0.498503	0.497161	0.494560	0.395040	0.391036	38533321
11	0.674304	0.674247	0.673980	0.534266	0.533288	0.531178	0.433048	0.430050	42526382
12	0.699097	0.699056	0.698844	0.565457	0.564729	0.562998	0.466919	0.464637	46059907
13	0.720420	0.720391	0.720220	0.592859	0.592307	0.590872	0.497216	0.495451	49202562
14	0.738947	0.738925	0.738786	0.617098	0.616671	0.615470	0.524427	0.523042	52011616
15	0.755189	0.755172	0.755057	0.638674	0.638339	0.637324	0.548967	0.547866	54535109
20	0.813374	0.813369	0.813317	0.718456	0.718336	0.717847	0.642223	0.641811	64052864
25	0.849270	0.849268	0.849241	0.769596	0.769543	0.769272	0.703997	0.703809	70307431
30	0.873605	0.873604	0.873588	0.805089	0.805062	0.804897	0.747748	0.747650	74719221
35	0.891182	0.891182	0.891172	0.831139	0.831124	0.831016	0.7803036	0.780247	77994340
40	0.904472	0.904472	0.904465	0.851062	0.851053	0.850979	0.805451	0.805416	80520467
45	0.914870	0.914870	0.914865	0.866788	0.866782	0.866729	0.825451	0.825429	82527551
50	0.923229	0.923229	0.923225	0.879515	0.879511	0.879471	0.841734	0.841718	84160327
55	0.930093	0.930093	0.930090	0.890025	0.890022	0.889992	0.855244	0.855233	85514592
60	0.935831	0.935831	0.935829	0.898850	0.898848	0.898824	0.866633	0.866625	86655607
65	0.940699	0.940699	0.940697	0.906365	0.906363	0.906345	0.876363	0.876357	87630219
70	0.944880	0.944880	0.944879	0.912841	0.912840	0.912825	0.884772	0.884767	88472326
75	0.948511	0.948511	0.948510	0.918480	0.918479	0.918467	0.892111	0.892107	89207130
80	0.951693	0.951693	0.951692	0.923435	0.923434	0.923424	0.898572	0.898569	89853961
85	0.954505	0.954505	0.954504	0.927821	0.927821	0.927812	0.904304	0.904302	90427698
90	0.957007	0.957007	0.957007	0.931733	0.931732	0.931725	0.909423	0.909421	90939982
95	0.959249	0.959249	0.959248	0.935242	0.935242	0.935236	0.914022	0.914021	91400227
100	0.961268	0.961268	0.961268	0.938409	0.938408	0.938403	0.918178	0.918176	91816023
105	0.963097	0.963097	0.963097	0.941280	0.941280	0.941275	0.921950	0.921949	0.921935
110	0.964761	0.964761	0.964761	0.943896	0.943895	0.943891	0.925390	0.925389	0.925377
115	0.966281	0.966281	0.966281	0.946288	0.946288	0.946285	0.928540	0.928539	0.928528
120	0.967676	0.967676	0.967676	0.948485	0.948485	0.948482	0.931434	0.931434	0.931424
125	0.968960	0.968960	0.968959	0.950509	0.950509	0.950507	0.934104	0.934103	0.934095
130	0.970145	0.970145	0.970145	0.952381	0.952381	0.952378	0.936573	0.936573	0.936565
135	0.971244	0.971244	0.971244	0.954116	0.954116	0.954113	0.938864	0.938864	0.938857
140	0.959989	0.972264	0.972264	0.955729	0.955729	0.955727	0.940996	0.940995	0.940989

Bibliography

1. Abdel-Ghaly, A. and others. "Evaluation of Competing Software Reliability Predictions," *IEEE Transactions on Software Engineering*, *12*(9):950–967 (September 1986).
2. Abdel-Ghaly, A. A. *Analysis of Predictive Quality of Software Reliability Models*. PhD dissertation, City University, London, U.K., 1986.
3. Akaiki, H. "A New Look at Statistical Model Identification," *IEEE Transactions on Automatic Control*, *19*:716–723 (1974).
4. Akaiki, H. *Prediction and Entropy*. Mathematics Research Center Technical Summary Report 2397, Madison, Wisconsin: University of Wisconsin Madison, 1982.
5. Anderson, T. W. *Introduction to Multivariate Statistical Analysis* (Second Edition). New York, NY: John Wiley and Sons, 1984.
6. Arlat, J. and Others. "Dependability Modeling and Evaluation of Software Fault-Tolerant Systems," *IEEE Transactions on Computers*, *39*:504–513 (1990).
7. Bailey, C. T. and W. L. Dingee. "A Software Study Using Halstead Metrics," *ACM/Sigmetrics*, *10*:189–197 (1981).
8. Balakrishnan, M. and C. S. Raghavendra. "On Reliability Modeling of Closed Fault-Tolerant Computer Systems," *IEEE Transactions on Computers*, *39*:571–575 (1990).
9. Barr, Donald R. and Peter W. Zehna. *Probability: Modeling Uncertainty*. Menlo Park, CA: Addison-Wesley, 1983.
10. Bastani, F. B. and C. V. Ramamoorthy. "Input-Domain Base Models for Estimating the Correctness of Process Control Programs." *Reliability Theory* edited by A. Serra and R. E. Barlow, 1321–1378, Amsterdam: North-Holland, 1986.
11. Belli, F. "Fault-Tolerant Programs and Their Reliability," *IEEE Transactions on Reliability*, *39*:184–192 (1990).
12. Blumenthal, S. and R. Marcus. "Estimating Population Size with Exponential Failure," *American Statistical Association Journal*, *70*:913–922 (December 1975).
13. Braun, H. and J. M. Paine. *A Comparative Study of Models for Reliability Growth*. Technical Report, Department of Statistics - Princeton University, 1977 (126 Series 2).
14. Brocklehurst, Sarah and others. "Recalibrating Software Reliability Models," *IEEE Transactions on Software Engineering*, *16*(4):458–470 (April 1990).
15. Brown, J. R. and M. Lipow. "Tesing for Software Reliability," *Proceedings of the International Conference on Reliable Software*. 518–527. April 1975.
16. Cote, V. and Others. "Software Metrics: An Overview of Recent Results," *J. of Systems and Software*, *8*:121–131 (1988).
17. Crow, L. H. "Reliability Analysis for Complex, Repairable Systems." *Reliability and Biometry* edited by F. Proshan and R. J. Serfling, 379–410, Philadelphia, PA: SIAM, 1974.
18. Crow, L. H. *Confidence Interval Procedures for Reliability Growth Analysis*. Technical Report, Aberdeen, MD: US Army Material Systems Analysis Activity, 1977 (197).
19. Crow, L. H. and N. D. Singpurwalla. "An empirically Developed Fourier Series Model for Describing Software Failures," *IEEE Transactions on Reliability*, *33*:176–183 (1984).

20. Davis, J. S. and R. J. LeBlanc. "A Study of Applicability of Complexity Metrics," *IEEE Transactions on Software Engineering*, *14*:1366–1372 (1988).

21. Dawid, A. P. *Calibration Based Empirical Probability*. Res. Report 36, London: Department of Statistical Science; University College, 1984.

22. Dawid, A. P. "Statistical Theory - The Prequential Approach," *Royal Statis. Soc.*, *147*:278–292 (1984).

23. DeGroot, Morris H. *Probability and Statistics* (First Edition). Menlo Park, CA: Addison-Wesley, 1975.

24. Dhillon, B. S. *Reliability in Computer System Design*. Norwood, NJ: Ablex Publishing Corp., 1987.

25. Dijkstra, E. W. "The Humble Programmer," *Communs ACM*, 859–866 (1972).

26. Down, T. and P. Garrone. "Some New Models of Software Testing with Performance Comparisons," *IEEE Transactions on Reliability*, *40*(3):322–328 (1991).

27. Duane, J. T. "Learning Curve Approach to Reliability Monitoring," *IEEE Transactions on Aerospace*, *2*:563566 (1964).

28. Forman, E. H. and N. D. Singpurwalla. "An Empirical Stopping Rule for Debugging and Testing Computer Software," *American Statistical Association Journal*, *72*:750–757 (1977).

29. Gail, M. H and J. L. Gastwirth. "A Scale Free Goodness-of-Fit Test for the Exponential Distribution based on the Gini Statistic," *Royal Statistical Society Series B*, *40*(3):350–357 (1978).

30. Goel, A. L. "Software Reliability Models: Assumptions, Limitations, and Applicability," *IEEE Transactions on Software Engineering*, *11*(12):1411–1423 (December 1985).

31. Goel, A. L. and K. Okumoto. "An Analysis of Recurrent Software Errors in a Real-Time Control System." *Proceedings of the ACM Conference*. 496–501. 1978.

32. Goel, A. L. and K. Okumoto. *An Imperfect Debugging Model for Software Reliability*. Syracuse University, Final Technical Report Vol. 1 RADC-TR-87-155, Rome Air Development Center (RADC) Griffis AFB NY, 1978.

33. Goel, A. L. and K. Okumoto. "Time Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," *IEEE Transactions on Reliability*, *28*(3):206–211 (August 1979).

34. Goseva-Popstojanova, K. D. and A. L. Grnarov. "A New Markov Model of N version Programming System," *Proceedings of the International Symp. on Software Reliability Engineering*, 210–217 (1991).

35. Hogg, Robert V. and Elliot A. Tanis. "An Iterative Procedure for Testing the Equality of Several Exponential Distributions," *American Statistical Association Journal*, *58*:435–443 (June 1963).

36. Hudson, G. R. *Programming Errors as a Birth-Death Process*. Technical Report SP-3011, System Development Corp., 1967.

37. Iannino, A. and Others. "Criteria for Software Reliability Model Comparisons," *IEEE Transactions on Software Engineering*, *10*(9):687–691 (November 1990).

38. Jelinski, Z. and P. B. Moranda. "Software Reliability Research." *Statistical Computer Performance Evaluation* edited by W. Freiberger, 465–484, New York, NY: Academic Press, 1972.

39. K. N. Cole, B. N. Nagarsenker and P. B. Nagarsenker. "A Test for Equality of Exponential Distributions Based on Type *II* Censored Samples," *IEEE Transactions on Reliability*, *36*:94-97 (1987).

40. Kapur, K. C. and L. R. Lamberson. *Reliability in Engineering Design*. New York, NY: John Wiley and Sons, 1977.

41. Keiler, P. A. and others. "Comparison of Software Reliability Predictions." *Dig. FTCS (13 th Int. Symp. Fault-Tolerant Comput.*, 128–134 (1983).

42. Keiler, P. A. and others. "On the Quality of Software Reliability Predictions." *Electronic Systems Effectiveness and Life Cycle Costing NATO ASI Series F3* edited by J. K. Skwirzynski, 441-460, New York, NY: Springer-Verlag, 1983.

43. Khoshgoftaar, T. M. "On Model Selection in Software Reliability," *Proceedings of the* 8^{th} *Symp. on Computational Statistics*, 13–14 (1988).

44. Khoshgoftaar, T. M. and T. G. Woodcock. "A Simulation Study of the Performance of the Akaike Information Criterion for the Selection of Software Reliability Growth Models," *Proceedings of the* 27^{th} *Annual South East Region ACM Conference*, 419–423 (1989).

45. Khoshgoftaar, T. M. and T. G. Woodcock. "Software Reliability Model Selection: A Case Study," *Proceedings of the International Symp. on Software Reliability Engineering*, 183–191 (1991).

46. Kline, M. B. "Software and Hardware R&M: What are the Differences ?." *Proceedings of the IEEE Annual Reliability and Maintainability Symposium*. 179–185. 1980.

47. Lawless, J. F. *Statistical Models and Methods for Lifetime Data*. New York, NY: John Wiley and Sons, 1982.

48. Lipow, M. *Estimation of Software Package Residual Errors*. Software Series Report TRW-SS-72-09, TRW Rodondo Beach , CA, 1972.

49. Littlewood, B. "How to Measure Software Reliability and How Not to," *IEEE Transactions on Reliability*, *28*:103–110 (1979).

50. Littlewood, B. "Theories of Software Reliability: How Good Are They and How Can They be Improved ?," *IEEE Transactions on Software Engineering*, *6*(5):489–500 (September 1980).

51. Littlewood, B. "Stochastic Reliability Growth: A model for fault-removal in computer programs and hardware designs," *IEEE Transactions on Reliability*, *30*:313–320 (1981).

52. Littlewood, B. "Forecasting Software Reliability." *Lectures Notes in Computer Science, Vol. 341 Software Reliability Modeling and Identification* edited by Sergio Bittani. 141–209, New York, NY: Springer-Verlag, 1988.

53. Littlewood, B. and A. Sofer. "A Bayesian Modification to the Jelinski-Moranda Software Reliability Growth Model," *Software Engineering Journal*, *2*:30–41 (1987).

54. Littlewood, B. and J. Verrall. "Likelihood Function of a Debugging Model for Computer Software Reliability," *IEEE Transactions on Reliability*, *30*:145–148 (1981).

55. Littlewood, B. and J. L. Verrall. "A Baysian Reliability Growth Model for Computer Software," *J. Roy. Statist. Soc. C.*, *22*:332–346 (1973).

56. Lyu, M. R. "Measuring Reliability of Embedded Software: An Empirical Study with JPL Project Data." *Proceedings of International Conference on Probabilistic Safety Assessment and Management*. 1991.

57. Lyu, M. R. and Allen Nikora. "A Heuristic Approach for Software Reliability Prediction: The Equality-Weighted Linear Combination Model," *Proceedings of the International Symp. on Software Reliability Engineering*, 172–181 (1991).

58. McCall and Others. *Methodology for Software Reliability Prediction Vol. II.* DTIC Report AD-A 190-019. Rome Air Development Center (RADC) Griffis AFB NY, 1987.

59. Mills, H. D. *On the Statistical Validation of Computer Programs.* Report FSC-72-6015, IBM Federal System Division Gaitherburg, MD, 1972.

60. Miyamoto, I. "Software Reliability in On-Line Real Time Environments." *Proceedings of the International Conference on Reliable Software*. 194–203. April 1975.

61. Moranda, P. B. "Predictions of Software Reliability During Debugging." *Proceedings of the IEEE Annual Reliability and Maintainability Symposium*. 327–332. 1975.

62. Munson, J. C. and T. M. Khoshgoftaar. "The Use of Software Complexity Metrics in Software Reliability Modeling," *Proceedings of the International Symp. on Software Reliability Engineering*, 2–11 (1991).

63. Musa, J. D. "A Theory of Software Reliability and its Application," *IEEE Transactions on Software Engineering*, *1*:312–327 (September 1975).

64. Musa, J. D. *Software Reliability Data.* report available from Data and Analysis Center for Software DACS, Rome Air Development Center (RADC) Griffis AFB NY, 1979.

65. Musa, J. D. "The Measurement and Management of Software Reliability." *Proc. IEEE,68(9)*. 1131–1143. 1980.

66. Musa, J. D. and K. Okumoto. "Software Reliability Models: Concepts, Classification, Comparisons, and Practice." *Electronic Systems Effectiveness and Life Cycle Costing NATO ASI Series F3* edited by J. K. Skwirzynski, 395–424, New York, NY: Springer-Verlag, 1983.

67. Musa, J. D. and K. Okumoto. "A Logarithmic Poisson Execution Time Model for Software Reliability Measurement." *Proceedings of the 7$^{\underline{th}}$ International Conference on Software Engineering*. 230–238. 1984.

68. Musa, J. D. and Others. *Software Reliability: Measurement, Prediction, Application*. New York, NY: McGraw-Hill, 1987.

69. Nagarsenker, B. N. and P. B. Nagarsenker. "On a Test of Equality of Two-Parameter Exponential Distributions," *Statistics and Probability Letters*, *2*:357–361 (1984).

70. Nagarsenker, B. N. and P. B. Nagarsenker. "Distribution of the LRT for Testing the Equality of several 2-Parameter Exponential Distributions," *IEEE Transactions on Reliability*, *34*:65–68 (1985).

71. Nagarsenker, B. N. and K. C. Pillai. "Distribution of the likelihood Ratio Criterion for Testing a Hypothesis Specifying a Covariance Matrix," *Biometrika*, *60*:359–364 (1973).

72. Nair, U. S. "Application of Factorial Series in the Study of Distribution laws in Statistics," *Sankhya*, *5*:175 (1940).

73. Nelson, E. "Estimating Software Reliability from Test Data," *Microelectronics and Reliability*, *17*:67–74 (1978).

74. Norlund, N. E. "Sur les series de facultes," *Acta Math*, *37*:327–387 (1914).

75. Ohba, M. and Xiao-Mei Chou. "Does Imperfect debugging Affect Software Reliability Growth ?." *Proceedings of the 11$^{\underline{th}}$ International IEEE Conference on Software Engineering*. 237–244. May 1989.

76. Ohba, M. and S. Yamada. "S-shaped Software Reliability Growth Curve: How Good is it?," *COMPSAC'82*, 38–44 (1982).

77. Okumoto, K. and A. L. Goel. "Optimum Release Time for Software Systems," *J. of Systems and Software*, *1*:315–318 (1980).

78. Ramamoorthy, C. V. and F. B. Bastani. "Software Reliability: Status and Perspective," *IEEE Transactions on Software Engineering*, 8:354–371 (1982).

79. Rosenblatt, M. "Remarks on a Multivariate Transformation." *Ann. Math Statist.*, *23*:470–742 (1952).

80. Schneidewind, N. F. "Analysis of Error Process in Computer Software." *Proceedings of the International Conference on Reliable Software*. 337–346. April 1975.

81. Shick, G. J. and R. W. Wolverton. "Assessment of Software Reliability." *Proc. Operation Research, Physica-Verlag* 395–422, Wurzburg-Wien, 1973.

82. Shick, G. J. and R. W. Wolverton. "An Analysis of Competing Software Reliability Models," *IEEE Transactions on Software Engineering*, *4*(5):104–120 (1978).

83. Shooman, M. L. "Probabilistic Models for Software Reliability Prediction." *Statistical Computer Performance Evaluation* edited by W. Freiberger, 485–502, New York, NY: Academic Press, 1972.

84. Shooman, M. L. "Software Reliability: A Historical Perspective," *IEEE Transactions on Reliability*, *33*:48–55 (1984).

85. Singpurwalla, N. D. and Refik Soyer. "Assessing(Software) Reliability Growth Using a Random Coefficient Autoregressive Process and Its Ramifications," *IEEE Transactions on Software Engineering*, *11*(12):1456–1464 (December 1985).

86. Sukert, A. "An Investigation of Software Reliability Models." *Proceedings of the Annual Reliability and Maintainability Symposium*. January 1977.

87. Thayer, T. A. and Others. *Software Reliability Study*. Final Technical Report RADC-TR-83-207, Rome Air Development Center (RADC) Griffis AFB NY, 1976.

88. Titchmarsh, E. C. *Introduction to the Theory of Fourier Integrals*. Oxford University Press, 1948.

89. Wagoner, W. L. *The Final Report of Software Reliability Measurement*. Aerospace Report TOR-0074(4112-1), Aerospace Corporation, August 1973.

90. Wilks, S. S. "Sample Criteria for Testing the Equality of Means, Equality of Variances, and Equality of Covariances in a Normal Multivariate Distribution," *Ann. Math Statist.*, *17*:257–281 (1946).

91. Xie, M. "A Shock Model for Software Failures," *Microelectronics and Reliability*, *27*:717–724 (1987).

92. Xie, M. *Software Reliability Modeling*. River Edge,, NJ: World Scientific Publishing, 1991.

93. Xizi, Huang. "The limit Condition of Some Time Between Failures Models of Software Reliability," *Microelectronics and Reliability*, *30*(3):481–485 (1990).

94. Yamada, S. and Others. "S-shaped Reliability Growth Modeling for Software Error Detection," *IEEE Transactions on Reliability*, *32*:475–478 (1983).

Vita

Salah Amin Elewa was born on 16 July 1954 in Ismailia, Egypt. He graduated from the Modern High School for Advanced Students in Ain Shams, Cairo in 1972 and joined the Military Technical College, Kobri El-Kobba, Cairo, from which he received the degree of Bachelor of Science in Electrical Engineering in July 1977. Upon graduation he served as a maintenance Engineer in the Egyptian Air Force. From July 1981 to February 1982, he was assigned to F-16 Training in Lackland AFB, San Antonio, Texas and Hill AFB, Utah. In 1984, he was assigned, as a part time student, to earn the Masters Degree in Electronics and Computers from Ain Shams University, Cairo, Egypt. Upon receiving his Master degree in December 1986, his assignment was to teach computer science in the Egyptian Air Academy. In August 1988, he was chosen by the Egyptian Department of Defense to earn a Doctorate Degree in Computer Science from the United States Air Force Institute of Technology (AFIT). He is a member of the IEEE Computer Society.

Permanent address: 7 Niazi St. off Farouk St.,
Zagazig, Egypt

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Sept 92	PhD Dissertation-Sept 92

4. TITLE AND SUBTITLE

DEVELOPMENT OF AN ENVIRONMENT FOR
SOFTWARE RELIABILITY MODEL SELECTION

5. FUNDING NUMBERS

6. AUTHOR(S)

Salah Amin Elewa, B.S,M.S.
Lieutenant Colonel, Egypt Air Force

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

School of Engineering
Air Force Institute of Technology
Wright-Patterson AFB, Ohio 45433

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/DS/ENC/92-1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

An environment was developed for solving the problem of selecting a proper software reliability model for a given set of software failures. The idea behind the environment developed in this dissertation was to liken a collected set of software failure data to a previous one that proved to fit well with a specified software reliability model. Software failures were assumed to have a two-Parameter exponential distribution with unequal type II censoring. A test criterion was derived for testing Ratio Criterion. The exact distribution of the test criterion was derived. An asymptotic approximation was also obtained and was found to be very close to the exact distribution when the number of failures were more than twenty. Software failure data, available from "Data and Analysis Center for Software (DACS)", were used as the initial group of software failure sets. The environment was then applied, for testing the quality of several software failure sets.

14. SUBJECT TERMS

SOFTWARE RELIABILITY MODELS, SOFTWARE FAILURES, MODEL SELECTION

15. NUMBER OF PAGES 222

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL